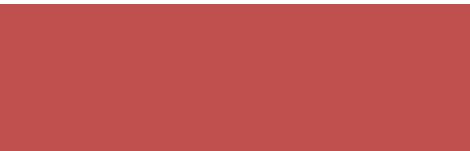
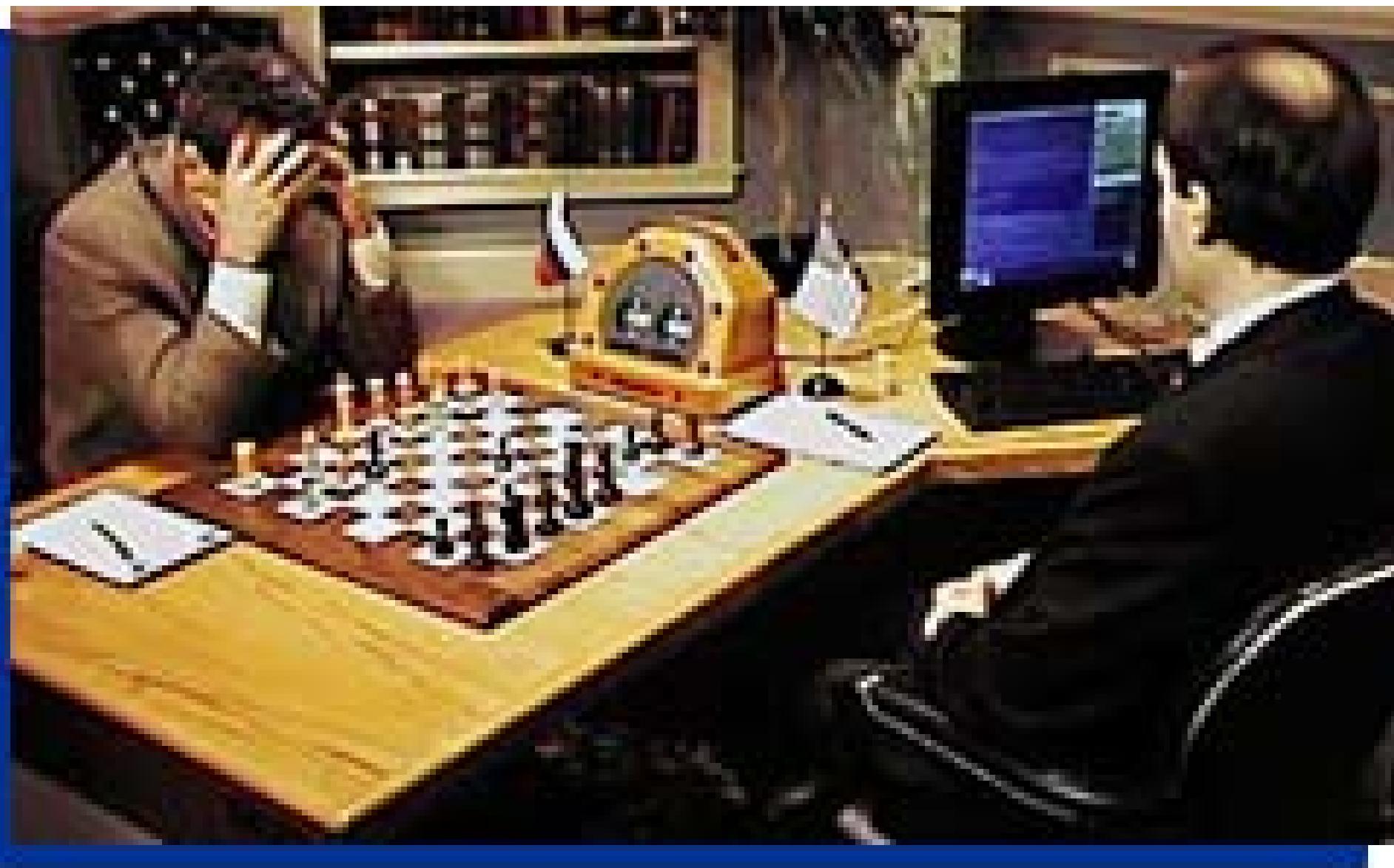


COMP 308
ARTIFICIAL INTELLIGENCE
READING - DEEP BLUE
DEFEATS KASPAROV

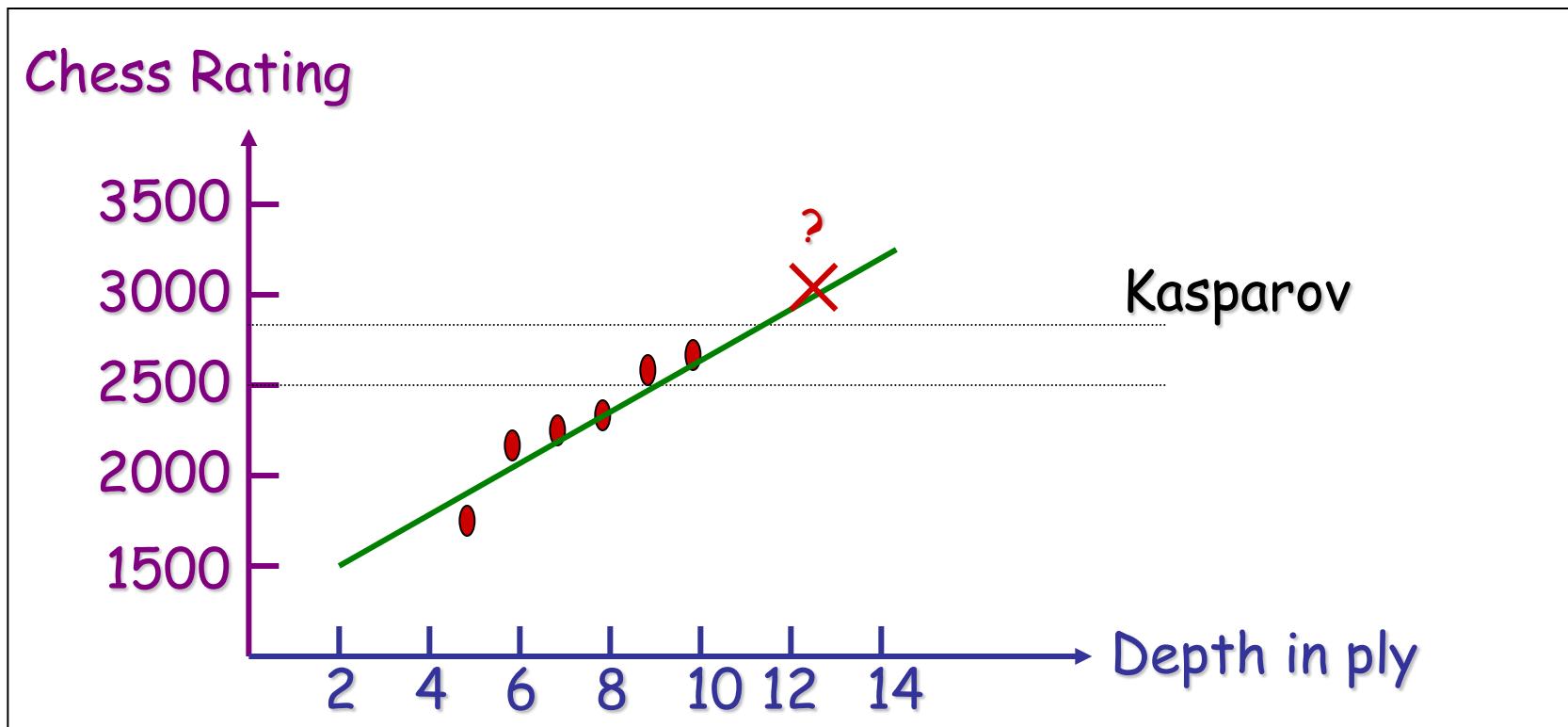




**Garry Kasparov and Deep Blue. © 1997,
GM Gabriel Schwartzman's Chess Camera, courtesy
IBM.**

Win of deep blue predicted:

Computer chess ratings studied around 90ies:



Further increase of depth was likely to win !

A week before his showdown with Deep Blue, Garry Kasparov sat down to discuss the significance of the rematch.

Q: What is at stake here? Is it about more than money?

GK: It's about the supremacy of human beings over machines in purely intellectual fields. It's about defending human superiority in an area that defines human beings.

Q: Why should people who aren't necessarily fans of chess be interested in the event?

GK: Because it tells us where we stand in a world of intelligent machines. There is always a deep fascination in watching a battle between two different and rival systems. When Fischer played against Spassky it was the free world against communism. People who knew nothing about chess were deeply concerned about the outcome. People are even more concerned about the results of the battle between man and machine.

Q: Please describe your training routine for this match. How long have you been preparing? Where? What does the training consist of?

GK: How long have I been training for this match? One could argue all of my life! Actually, I have done intense general chess training for the last two years and in the last four weeks I have trained specifically for this match against the computer. It was done in Podolsk, which is just outside Moscow. Interestingly, I make a lot of use of computers in my training. I think the other side works more intensely with human beings.

In a shocking finale that lasted barely more than an hour, World Champion Garry Kasparov resigned 19 moves into Game 6, handing a historic victory to Deep Blue.

The win gave the IBM supercomputer a 3.5-2.5 victory in the six-game rematch. It was the first time a current world champion has lost a match to a computer opponent under tournament conditions.

"What we just witnessed was a landmark achievement in chess," said match commentator Yasser Seirawan. "All I can say is that I'm stunned. I absolutely didn't expect this to happen."

Kasparov, who afterward admitted he was in a poor frame of mind entering Game 6, fell into a well-known trap that has been established as a bad line to follow. According to commentators covering the match, it was almost inconceivable that someone of Kasparov's ability could allow this to happen.

It appeared that the pressure of the past five games had taken its toll on Kasparov. His disastrous mistake early in the game was certainly uncharacteristic of a man generally considered to be the greatest player in the history, and Kasparov's early resignation was a sign that he'd lost his will to fight. "For me, the match was over yesterday," he said. "I had no real strength left to fight. And today's win by Deep Blue was justified."

Game 6 began with a very quiet, positional-based opening, the Caro-Kann. Unlike in four of the previous five games, Kasparov, playing black, began with a "real" opening — one that wasn't specifically improvised to throw off Deep Blue -- but he then developed it into a losing variation.

Based on this beginning, the commentators predicted a strategic battle. Said Seirawan, "This is also a very solid set up for black."

That forecast quickly went out the window after Kasparov's seventh move, when Deep Blue sacrificed a knight for a pawn, costing Kasparov the ability to castle. At this point, Kasparov's disposition changed dramatically. "Kasparov has a look of terror on his face," said Seirawan. "He's showing his disbelief by falling for a well-known opening trap." The error provided Deep Blue with a highly advantageous position and marked the beginning of the end for Kasparov.

After the game, Deep Blue development team leader C.J. Tan expressed satisfaction with the result. He also pointed out that there was more to Deep Blue's battle with Garry Kasparov than just a game of chess. "We are very proud to have played a role in this historic event," he said. "This will benefit everyone -- from the audience to school children, to businesses everywhere, even to Garry Kasparov."

Kasparov, who on several occasions expressed unhappiness with the ground rules of the six-game rematch, challenged Deep Blue to a showdown under regular tournament conditions. "The match was lost by the world champion," he said, "but there are very good and very profound reasons for this. I think the competition has just started. This is just the beginning."

The Deep Blue development team took home the \$700,000 first prize, while Kasparov received \$400,000.

By his own admission, the pressure got to Garry Kasparov today. It was the not the \$300,000 difference between first and second prize nor the massive media attention this match has received. It was Deep Blue's astonishing play the world champion could not come to terms with.

Kasparov was off-balance coming into this game, a man who, for once in his career, had let his emotions overcome the logical impulses of his own chess genius. Still smarting from his reverse in the second game, Kasparov had lost his objectivity and accepted his strategy has failed.

In a bizarre twist, Kasparov avoided his favorite opening moves and started to play like his longtime human rival, Anatoly Karpov, who loves to defend with an opening known as the Caro-Kann. This was a clear sign things were not right, but Deep Blue, naturally, did not notice and just played the standard moves. As early as move seven, Kasparov made a clear mistake allowing a sacrifice of a knight that is known to be very strong. A quick check of the [computer chess databases](#) showed that of the nine players who had allowed this sacrifice, only one had survived and he needed a large slice of luck.

With Deep Blue, luck does not come into it, and we witnessed the shortest ever game between man and machine at the top level. After just under an hour, Kasparov realized how hopeless his position had become. We did not have to wait long for the killer blow from Deep Blue that ended the game after just 19 moves and win the match 3.5-2.5.

The champion issued a challenge at the post game press conference: " It's time for Deep Blue to play real chess. I personally guarantee I will tear it in pieces." Fighting talk, and I fervently hope we will see Deep Blue participate in wider world of chess.

What has changed in the machine that lost last year ? The director of the IBM research team, C.J.Tan, explained: " Three things were improved this time around; it's more powerful, we added more chess knowledge and we developed a program to change the parameters in between each game. "

Kasparov is still in shock, and was in his hotel room last night studying printouts from the Deep Blue team that he hopes will give him some insights into its wonderful play that have entertained and will, in time, educate every chess player.

Vishwanathan Anand became India's first grandmaster in 1987. He met Garry Kasparov for the world championship in 1995 at the World Trade Center in New York. Kasparov retained the title, but Anand remains one of the world's elite chess players.

I eagerly waited to see the Kasparov vs. Deep Blue rematch. Deep Blue was stronger. Deeper, to be precise. From my own experience, practical play exposes all sorts of weaknesses and strengths in my play that are hidden during preparation. Equally, the team behind Deep Blue must have benefited immensely from studying the six games played against Kasparov in 1996. And it would be faster. I can't tell the difference between 100 zillion positions and 497 zillion positions, but if it helped Deep Blue play stronger, so be it. I was looking forward to Deep Blue boldly going where no man had gone before.

Kasparov himself must have studied the games played last year. However, humans can't change their style drastically like computers. On top of that, all his games were accessible to the Deep Blue team, while he was in the dark about Deep Blue. He had two options: to play like Kasparov or to play like "Mr. Anti Deep Blue." The former runs the risk of playing to the strengths of the machines, the latter that the human ends up as disoriented as the machine. Humans, too, play weaker in unfamiliar situations and though they may find their way around better, machines can compensate for that with brute force.

Kasparov chose the latter. Unfortunately, as a result, we were never able to see the fabulous calculating abilities of Deep Blue. Not once did we see a spectacular example of brute force producing a solution that differed significantly from that suggested by intuition. A lot has been made of Deep Blue's play in the second game, but in fact only one or two moments can be singled out - 26.f4 and 37.Bxe4. The rest of the game is not that difficult, even for a computer.

There is also the mystery at the end of the game. Did Deep Blue not see 45...Qe3? Why on Earth did it play 44.Kf1? Surely it could calculate 3 moves further!

His strategy might even have worked if he hadn't conceded so much territory to Deep Blue. By trying so hard to avoid any position where Deep Blue might be able to calculate its way through, he effectively self-destructed. Three tough draws followed where he was always better, but unable to overcome Deep Blue's stubborn defense. By the 6th game, he was a pale shadow of himself. Suffice it to say, that the trap he fell into in the 6th game is a well known one. It forms part of his own opening strategy as White!!

The chess may have been disappointing, but the media interest has been exceptional and that is a wonderful promotion for the game of chess

Deep Blue has only played twelve games in two years against one single opponent. As such, it is impossible to tell how strong it is or what it is capable of.

IBM can hardly risk the reputation of its "blue-eyed" baby against some PC or mere mortal. So the rest of us (6,000,000,000 minus Kasparov) are left with more questions than answers.

The following is a top ten listing of the dissimilarities between the way Garry Kasparov and Deep Blue play chess:

1. Deep Blue can examine and evaluate up to 200,000,000 chess positions per second

Garry Kasparov can examine and evaluate up to three chess positions per second

2. Deep Blue has a small amount of chess knowledge and an enormous amount of calculation ability.

Garry Kasparov has a large amount of chess knowledge and a somewhat smaller amount of calculation ability.

3. Garry Kasparov uses his tremendous sense of feeling and intuition to play world champion-calibre chess.

Deep Blue is a machine that is incapable of feeling or intuition.

4. Deep Blue has benefitted from the guidance of five [IBM research scientists](#) and one international grandmaster.

Garry Kasparov is guided by his coach Yuri Dokhoian and by his own driving passion play the finest chess in the world.

5. Garry Kasparov is able to learn and adapt very quickly from his own successes and mistakes.

Deep Blue, as it stands today, is not a "learning system." It is therefore not capable of utilizing artificial intelligence to either learn from its opponent or "think" about the current position of the chessboard.

6. Deep Blue can never forget, be distracted or feel intimidated by external forces (such as Kasparov's infamous "stare").

Garry Kasparov is an intense competitor, but he is still susceptible to human frailties such as fatigue, boredom and loss of concentration.

7. Deep Blue is stunningly effective at solving chess problems, but it is less "intelligent" than even the stupidest human.

Garry Kasparov is highly intelligent. He has authored three books, speaks a variety of languages, is active politically and is a regular guest speaker at international conferences.

8. Any changes in the way Deep Blue plays chess must be performed by the members of the development team between games.

Garry Kasparov can alter the way he plays at any time before, during, and/or after each game.

9. Garry Kasparov is skilled at evaluating his opponent, sensing their weaknesses, then taking advantage of those weaknesses.

While Deep Blue is quite adept at evaluating chess positions, it cannot evaluate its opponent's weaknesses.

10. Garry Kasparov is able to determine his next move by selectively searching through the possible positions.

Deep Blue must conduct a very thorough search into the possible positions to determine the most optimal move (which isn't so bad when you can search up to 200 million positions per second).

Interesting and useful links:

- Meet Meena, Google's new chatbot that interacts like humans

<https://www.indiatoday.in/technology/news/story/meet-meena-google-s-new-chatbot-that-interacts-like-humans-1641982-2020-01-31>

- Sophia the Robot

<https://www.youtube.com/watch?v=1noWqYaDpiQ>

- Bina48

<https://www.youtube.com/watch?v=mfcyq7uGbZg>

- Mars 2020: The Next Mission to Mars

<https://www.youtube.com/watch?v=iZCRFRgSgas>

- Audi recently demonstrated a car that can park itself without the need for a driver

<https://www.youtube.com/watch?v=vt20UnkmkLI>

https://www.youtube.com/watch?v=Qa_ZSRj0WM0

- Meet Valerie, Carnegie Mellon's First Storytelling Roboceptionist chatbot

http://www.cmu.edu/cmnews/extra/040219_valerie.html

<http://roboceptionist.org/project.htm>

- ELIZA the psychotherapist chatbot

<http://nlp-addiction.com/eliza/>

- A.L.I.C.E chatbot

www.alicebot.org

- Apple's Siri

<https://www.apple.com/ios/siri/>

- Google's Driverless car

<http://www.google.com/about/careers/lifeatgoogle/self-driving-car-test-steve-mahan.html>

- Google Goggles

<http://googlemobile.blogspot.com/2010/10/open-your-eyes-google-goggles-now.html>

- IBM Watson

<http://www.ibm.com/smarterplanet/us/en/ibmwatson/>

- IBM wants FDA to let Watson diagnose patients:

<http://www.dispatch.com/content/stories/business/2015/02/02/ibm-wants-fda-to-relax-onwatson.html>

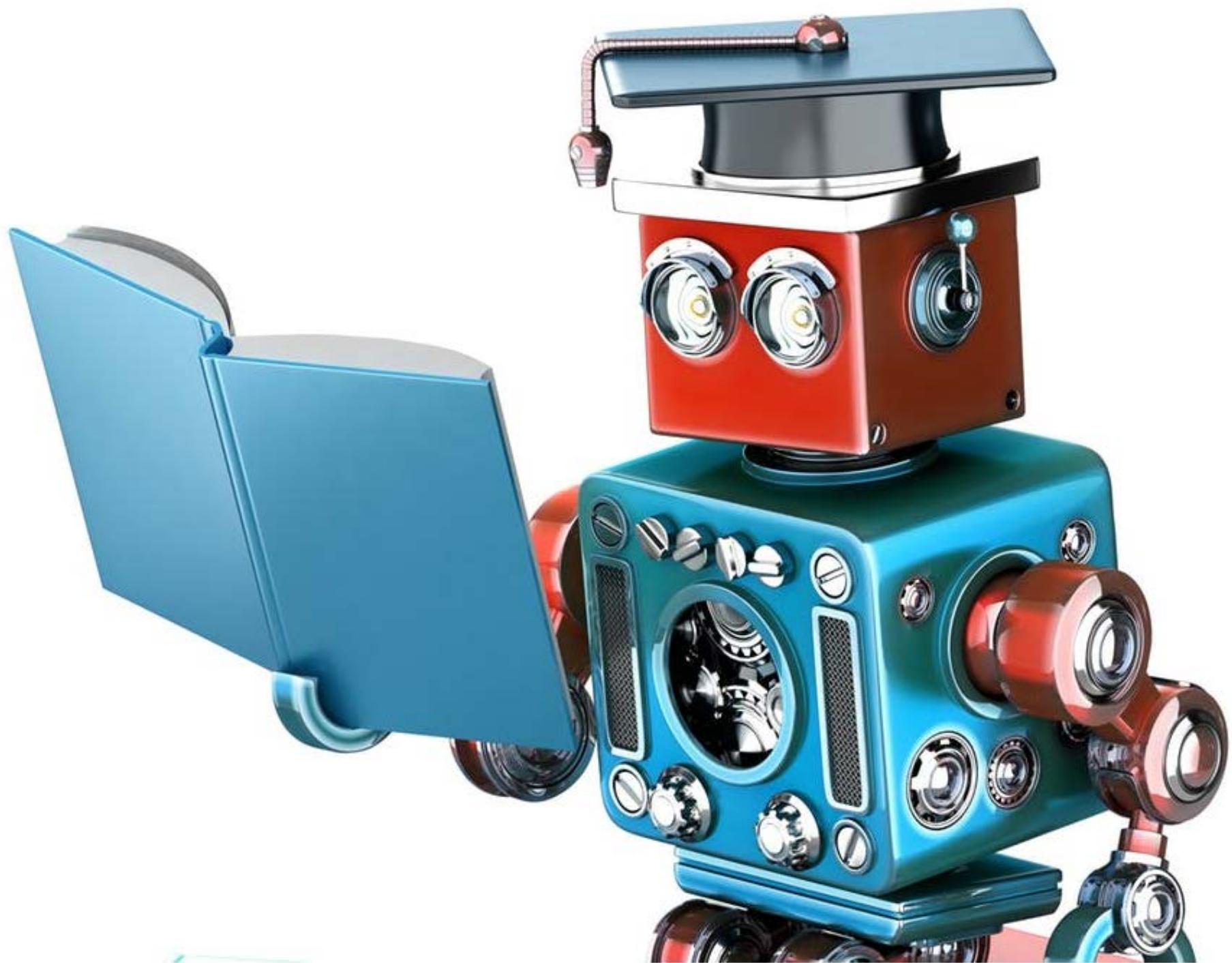
COMP 308

ARTIFICIAL INTELLIGENCE

PART 1 – INTRODUCTION

Njeri Ireri

Jan – April 2020



We Shall Discuss

- Intelligence
- What is A.I?
- Foundations of A.I - Multi-disciplinary domain
- A Brief History
- Approaches to A.I
- A.I Applications

Intelligence

- Dictionary definition
 - (1) : the ability to learn or understand or to deal with new or trying situations : **REASON**; also : the skilled use of reason
 - (2) : the ability to apply knowledge to manipulate one's environment or to think abstractly as measured by objective criteria (as tests)

-Merriam-Webster's Collegiate Dictionary

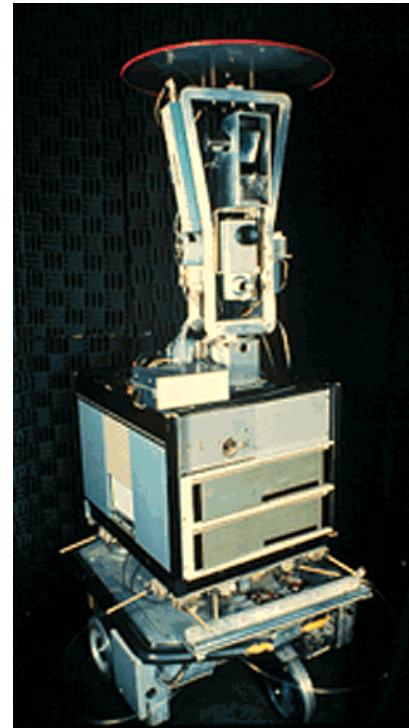
- Main ideas concerned with
 - Thought processes and reasoning
 - Behavior

Intelligence – 2 views

- **Intelligence = Autonomous movement**
- Intelligence = Thinking



Vaucanson
18th century



Shakey (1970)
Stanford Research Institute



SONY Aibo (1998)

Intelligence – 2 views

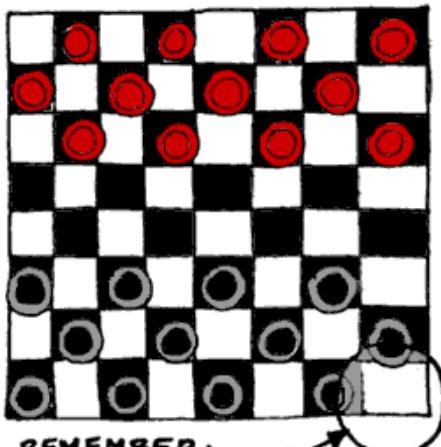
- Intelligence = Autonomous movement
- **Intelligence = Thinking**



Deep Blue defeats Garry Kasparov (1997)

Playing Games: Draughts

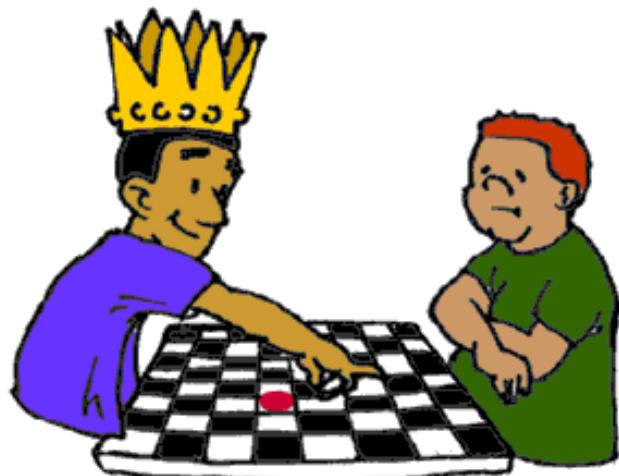
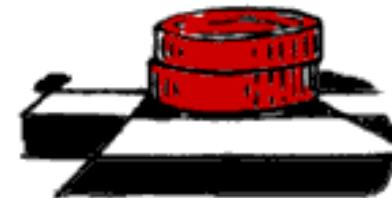
THE SET-UP



THE CAPTURE

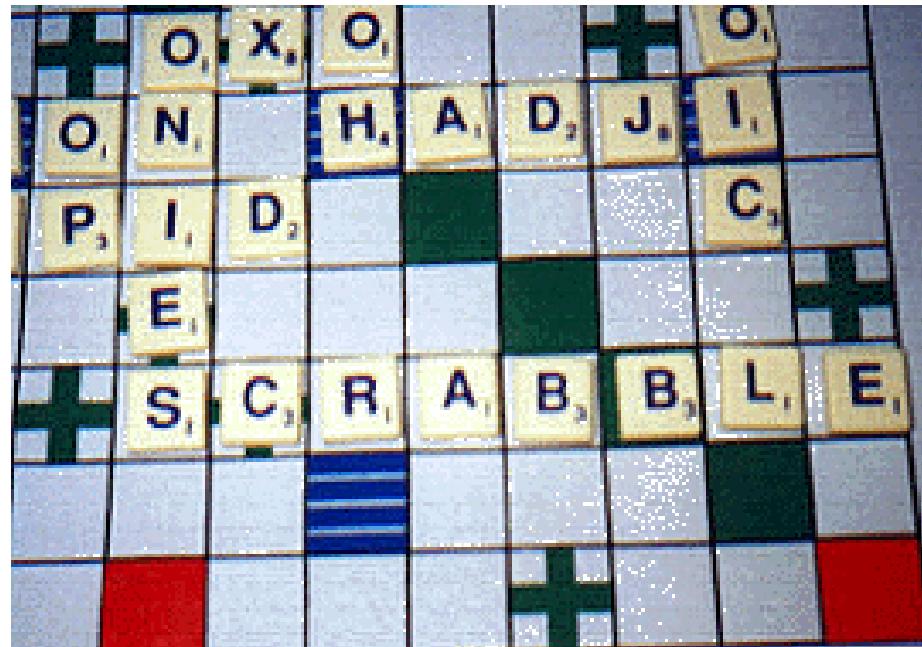


"CROWNED"



Is a computer more intelligent
if it beats you in a game of draughts?

Playing Games: Scrabble



Is a computer more intelligent
if it beats you in a game of scrabble?

Playing Games: Jigsaw



Is a computer more intelligent if it solves the puzzle faster than you do?

Finding Your Way: Maps



Travel from one town or village to another within shortest time and cheapest;
e.g. Cyangugu to Kigarama

Intelligence

- Types of intelligence (Multiple intelligence theory, Howard Gardner)
 - Linguistic-verbal intelligence
 - Logical-Mathematical intelligence
 - Musical intelligence
 - Spatial intelligence
 - Intrapersonal intelligence
 - Interpersonal intelligence
 - Naturalist intelligence
 - Bodily-Kinesthetic intelligence
 - "What makes life interesting, however, is that we don't have the same strength in each intelligence area, and we don't have the same amalgam of intelligences. Just as we look different from one another and have different kinds of personalities, we also have different kinds of minds."
- 
- Theoretical foundations
for recognizing different
talents and abilities in
people

Defining Artificial Intelligence

- There is no agreed definition of the term artificial intelligence. However, there are various definitions that have been proposed. These are considered below.
 - AI is a study in which computer systems are made that **think like human beings**. Haugeland, 1985 & Bellman, 1978.
 - AI is a study in which computer systems are made that **act like people**. AI is the art of creating computers that **perform functions that require intelligence when performed by people**. Kurzweil, 1990.
 - AI is the study of how to make computers **do things which at the moment people are better at**. Rich & Knight
 - AI is a study in which **computers that rationally think** are made. Charniac & McDermott, 1985.
 - AI is the study of **computations that make it possible to perceive, reason and act**. Winston, 1992

Defining Artificial Intelligence

- AI is the study in which **systems that rationally act** are made. AI is considered to be a study that **seeks to explain and emulate intelligent behaviour in terms of computational processes.** Schalkeoff, 1990.
- AI is considered to be a branch of computer science that is concerned with the **automation of intelligent behavior.** Luger & Stubblefield, 1993.
- ...
- Main ideas concerned with
 - Thought processes and reasoning
 - Behavior

Defining Artificial Intelligence

- Based on different definitions by different authors, the definitions of AI can be organized into 4 main categories

A system that can:

- Think like humans
- Act like humans
- Think rationally
- Act rationally



Defining Artificial Intelligence

	“Like People”	“Rationally”
Think	Cognitive Science	Laws of Thought
Act	Turing Test	Rational Agents

What is A.I?

- ‘**Systems that think like humans**’:
 - The exciting new effort to make computers think ... machines with minds
 - The automation of activities that we associate with human thinking e.g. decision making, problem solving, learning etc.
- Thinking humanly: Cognitive Modeling
 - Goal: Develop precise theories of human thinking
 - We need a way to determine how we think:
 - Through introspection
 - Through psychological experiments
 - How to validate? Requires
 - Predicting and testing behavior of human subjects (topdown)
 - Direct identification from neurological data (bottom-up)
 - Cognitive Sciences now distinct from AI

What is A.I?

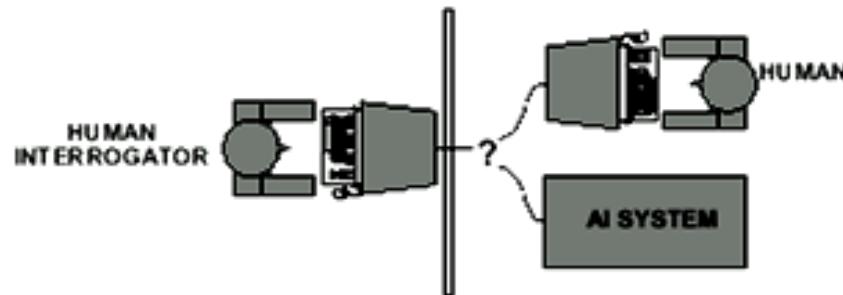
- Thinking humanly: Cognitive Modeling
 - **Problem:**
 - Identifiability: It may be impossible to identify the detailed structure of human problem solving using only externally available data.
 - “Optimal” performance is an excellent predictor of human performance in most routine tasks.

What is A.I?

- ‘**Systems that act like humans**’:
 - The art of making machines that perform functions that require intelligence when performed by people
 - The study of how to make computers do things which, at the moment, people do better
- Acting humanly: Turing Test
 - Alan Turing (1950) "Computing machinery and intelligence":
 - “Can machines think?” \leftrightarrow “Can machines behave intelligently?”
 - Can Computer fool a human interrogator?

What is A.I.?

- Acting humanly: Turing Test



- Anticipated all major arguments against AI in following 50 years
- Suggested major components of AI:
 - Natural language processing
 - Knowledge representation
 - Automated reasoning
 - Machine learning

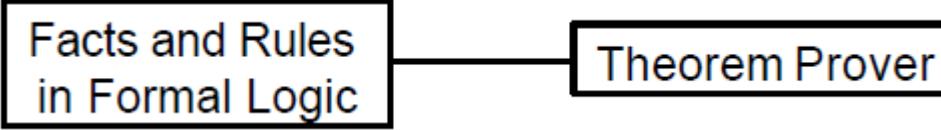
What is A.I?

- Acting humanly: Turing Test
 - **Problems:**
 - Tends to focus on human-like errors, linguistic tricks, etc.
Does not produce the most useful computer programs
 - Turing test is not reproducible, constructive, and amenable to mathematic analysis
 - What about physical interaction with interrogator and environment?
 - **Total Turing Test:** Requires physical interaction and needs perception (vision) and actuation (robotics)

What is A.I?

- ‘**Systems that think rationally**’:
 - Study of mental faculties through the use of computational models
 - Study of the computations that make it possible to perceive, reason and act
- Thinking Rationally: “Laws of thought” (The Logical Approach)
 - Aristotle: what are correct arguments/thought processes?
 - Several Greek schools developed various forms of *logic*: *notation* and *rules of derivation* for thoughts; may or may not have proceeded to the idea of mechanization
 - Direct line through mathematics and philosophy to modern AI

What is A.I?

- Thinking Rationally: “Laws of thought”
 - Ensure that all actions performed by computer are justifiable (“rational”)

```
graph LR; A["Facts and Rules  
in Formal Logic"] --- B["Theorem Prover"]
```
 - Rational = Conclusions are provable from inputs and prior knowledge
- Problems:
 - Not all intelligent behavior is mediated by logical deliberation- Representation of informal knowledge is difficult
 - Hard to define “provable” plausible reasoning
 - Combinatorial explosion: Not enough time or space to prove desired conclusions
 - What is the purpose of thinking? What thoughts should I have?

What is A.I?

- ‘**Systems that act rationally**’:
 - The study and design of intelligent agents
 - AI is concerned with intelligent behavior artifacts
- Acting rationally: rational agent
 - Rational behavior = the thing that which is expected to maximize goal achievement, given the available information
 - Does acting rationally have to necessarily involve thinking?
 - blinking reflex
 - or reflexive withdrawal of hand from hot oven
 - Not necessary ... but thinking should be in the service of rational action

What is A.I.?

- Acting rationally: rational agent
 - **Claim:** “Rational” means more than just logically justified. It also means “doing the right thing”

Rational agents do the best they can
given their resources

very few resources

no thought

“reflexes”

lots of resources

limited,
approximate
reasoning

Careful, deliberate
reasoning

- Adjust amount of reasoning according to available resources and importance of the result
- This is one thing that makes AI hard

So, Why Study A.I?

- Now that we know what A.I is, why is it exciting?
 - tries to understand intelligent entities, therefore we learn more about ourselves
 - strives to build intelligent entities, these are useful and interesting e.g. robots,
 - still a fairly new discipline and has lots of opportunities, good ideas have not already been taken by Galileo, Newton, Einstein and the rest. There's still openings for a full-time Einstein
 - regularly cited as the “*field I would most like to be in*” by scientists in other disciplines

..but the Objectives of A.I

- Understand how living beings behave, think, learn
- Engage in experiments by building artificial systems
- Derive applications for robotics and computers

Foundations of AI, a Multi-disciplinary Domain

- i.e. the main disciplines that have contributed ideas viewpoints and techniques that we find useful in AI
- Philosophy:
 - Can formal rules be used to draw valid conclusions?
 - How does the mental mind arise from a physical brain?
 - Where does knowledge come from?
 - How does knowledge lead to action?
- Mathematics; algorithms; formal representation and proof
 - What are the formal rules to draw valid conclusions?
 - What can be computed?
 - How do we reason with uncertain information?
- Computer Engineering
 - How can we build an efficient computer?

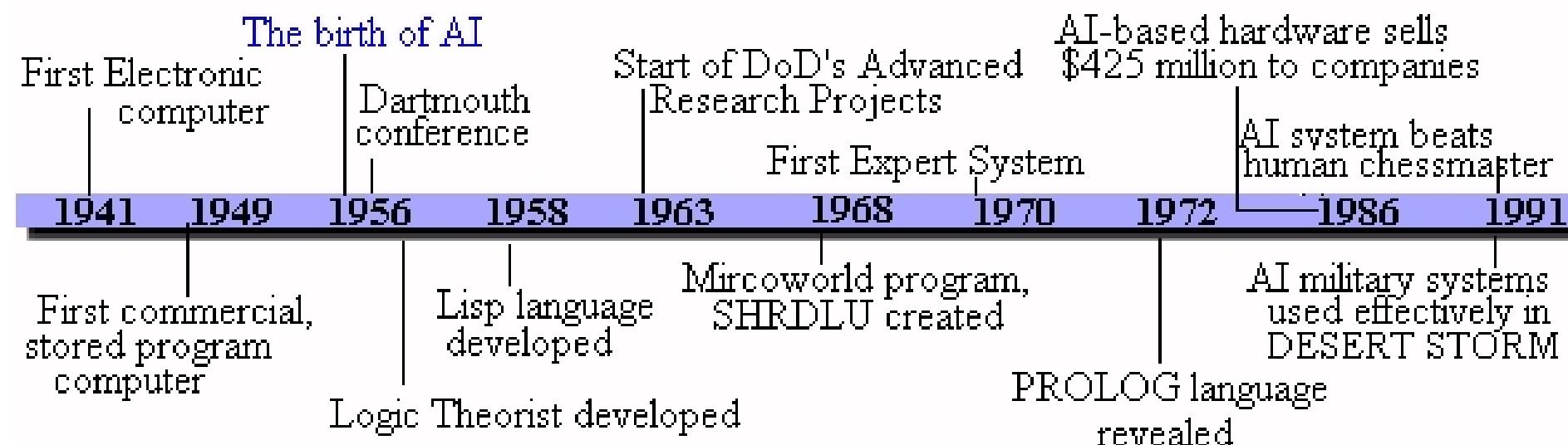
Foundations of AI, a Multi-disciplinary Domain

- Economics: Utility, decision theory
 - How should we make decisions so as to maximize payoff?
 - How should we do this when others may not go along?
- Neuroscience:
 - How do brains process information?
- Psychology
 - How do humans and animals think and act?
- Control theory and cybernetics:
 - How can artifacts operate under their own control?
- Linguistics:
 - How does language relate to thought?

History of A.I?

- ‘Gestation of AI’ (1943-1955)
 - 1943 McCulloch & Pitts: Boolean circuit model of brain
 - 1950 Turing’s “Computing Machinery and Intelligence”
 - 1950s Early AI programs, including Samuel’s checkers (draughts) program
 - Newell & Simon’s Logic Theorist, Gelernter’s Geometry Engine
- 1956 Birth of Artificial Intelligence - Dartmouth meeting: “Artificial Intelligence” adopted
- 1966–74 A dose of reality - AI discovers computational complexity, Neural network research almost disappears
- 1969–79 Early development of knowledge-based systems
- 1980–88 Expert systems industry booms
- 1988–93 Expert systems industry busts: “AI Winter”
- 1985–95 Neural networks return to popularity
- AI becomes a science (1987 – Present)
- Emergence of intelligent agents (1995 – Present)

History of A.I?



A.I: Evaluation

- AI has tended to succeed more in expert tasks (theorem proving, game playing, medical diagnosis)
- Than in mundane tasks (route planning, recognizing people and objects, communicating in natural language, navigating round objects etc.)

Approaches to A.I

- Strong AI
 - Aim: to build machines that can truly reason and solve problems; machines that are self aware and whose overall intellectual ability is indistinguishable from that of a human
 - Products not yet realized
 - The **Chinese room experiment** by John Searle refutes the Strong AI (Turing Test) claim that:
 - “The appropriately programmed computer with the right inputs and outputs would thereby have a mind in exactly the same sense human beings have minds”
- Weak AI
 - Aim: create some form of AI that cannot truly reason and solve problems ... but can act as if intelligent; machines that simulate human cognition
 - Some success has been reported in this area

The Chinese Room Experiment

- It is a thought-experiment first published in 1980 by American philosopher John Searle
- Searle imagines himself alone in a room following a computer program for responding to Chinese characters that are slipped to him under the door. Searle understands nothing of Chinese, and yet, by following the program for manipulating symbols and numerals just as a computer does, he produces appropriate strings of Chinese characters that fool those outside into thinking there is a Chinese speaker in the room
- The question Searle wants to answer is this: does the machine *literally* "understand" Chinese? Or is it merely *simulating* the ability to understand Chinese? Searle calls the first position "strong AI" and the latter "weak AI" approach

The Chinese Room Experiment

- The narrow conclusion of the argument is that programming a digital computer may make it appear to understand language but does not produce real understanding. Hence the “Turing Test” is inadequate. Searle argues that the thought experiment underscores the fact that computers merely use syntactic rules to manipulate symbol strings, but have no understanding of meaning or semantics.
- The broader conclusion of the argument is that the theory that human minds are computer-like computational or information processing systems is refuted. Instead minds must result from biological processes; computers can at best simulate these biological processes

Approaches to A.I

- Applied AI
 - Aim: produce commercially viable ‘smart’ systems e.g. a security system able to recognize faces in order to facilitate or deny access to premises
 - Considerably successful
- Cognitive AI
 - Aim: to use computers to test theories on how the human mind works e.g. theories on how we recognize faces, objects etc.

Basic Human Intelligent Behavior

- Perception (Seeing, hearing)
- Reasoning
- Learning
- Understanding Language
- Solving Problems

A.I Application Areas

- General problem solving
 - machine learning, automated reasoning, optimization, resource allocation, planning, scheduling, real-time problem solving, intelligent assistants, internet agents
- Computer Vision and Image recognition
 - Understanding images
- Robotics
 - engineering physical movement
- Natural language processing Natural language understanding
 - understanding written text
- Speech Recognition
 - understanding human speech
- Knowledge Based Systems
 - model human expertise in a limited area

Key Study Areas in A.I and

- Knowledge Based Systems
 - model human expertise in a limited area
- Reasoning
 - Uncertainty: probabilistic approaches
 - Decision theory
- Machine Learning
- Perception
 - Vision, natural language, robotics
- General Algorithms
 - Search, planning, constraint satisfaction
- Applications:
 - Game playing, AI in education, distributed agents

A.I applications – Sample Landmark Projects

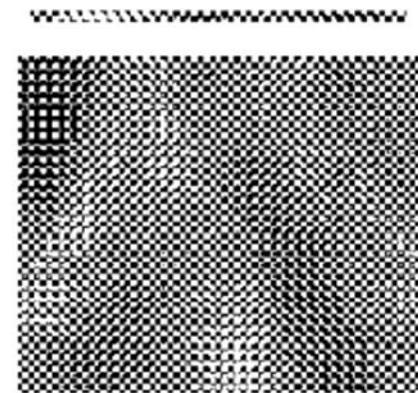
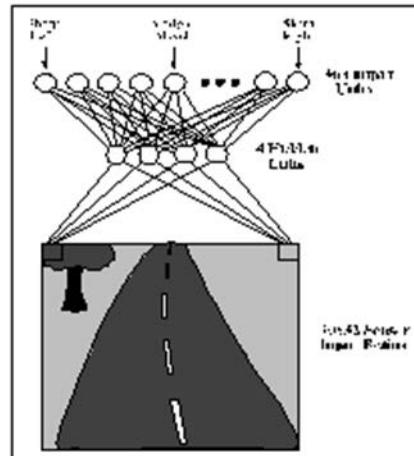
- MYCIN
- PROSPECTOR
- Autonomous Land Vehicle in a Neural Network (ALVINN)
- Deep Blue
- Machine Translator Programs
- Robotics
- Data Mining
- ...

Knowledge-based systems

- Medical Diagnosis - MYCIN
 - ▣ 1971, A program that could diagnose blood infections. It had 450 rules
 - ▣ It identifies bacteria causing severe infections, such as bacteremia and meningitis, and recommends antibiotics
- Mineral Prospecting - PROSPECTOR
 - ▣ 1979, A program that with geological data. It recommended exploratory drilling sites that proved to have substantial molybdenum deposits

ALVINN

- The Autonomous Land Vehicle in a Neural Network is a perception system which learns to control the NAVLAB vehicles by watching a person drive
 - Created in 1989
 - Drove itself some 2850 miles across the US
 - Drives 70 mph on a public highway
 - 30 outputs for steering
 - 4 hidden units
 - 30x32 pixels as inputs
 - 30x32 weights into one out of four hidden unit



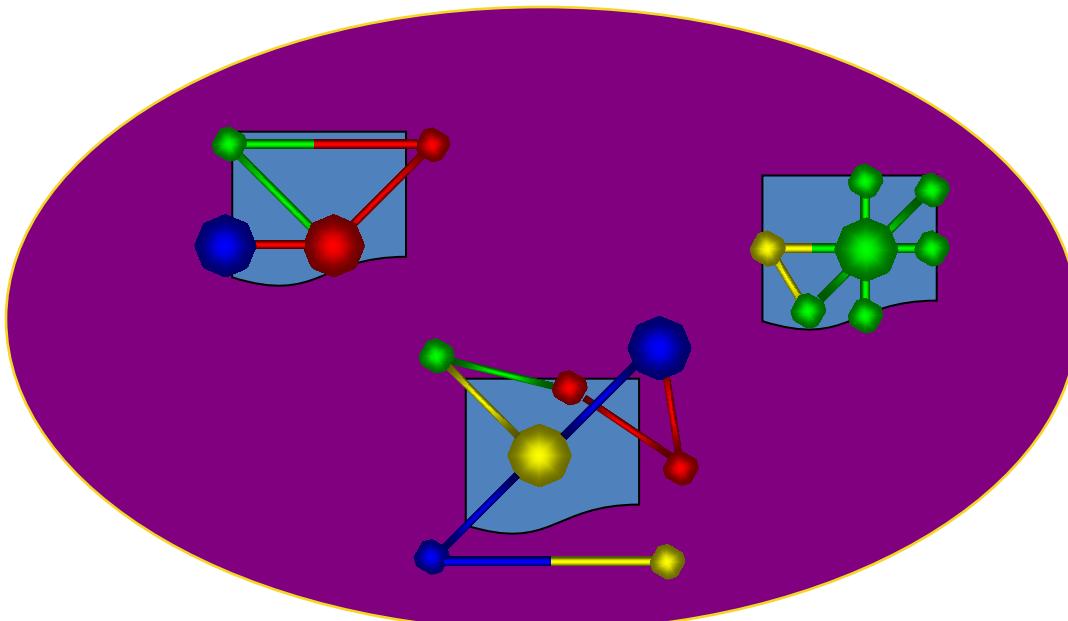
Machine Translator Programs

- Phraselator
 - Used by the US Military in the war in Iraq to quiz POWs, injured, communicate at checkpoints etc.

- Speechlator
 - For use in doctor-patient interviews

Data Mining

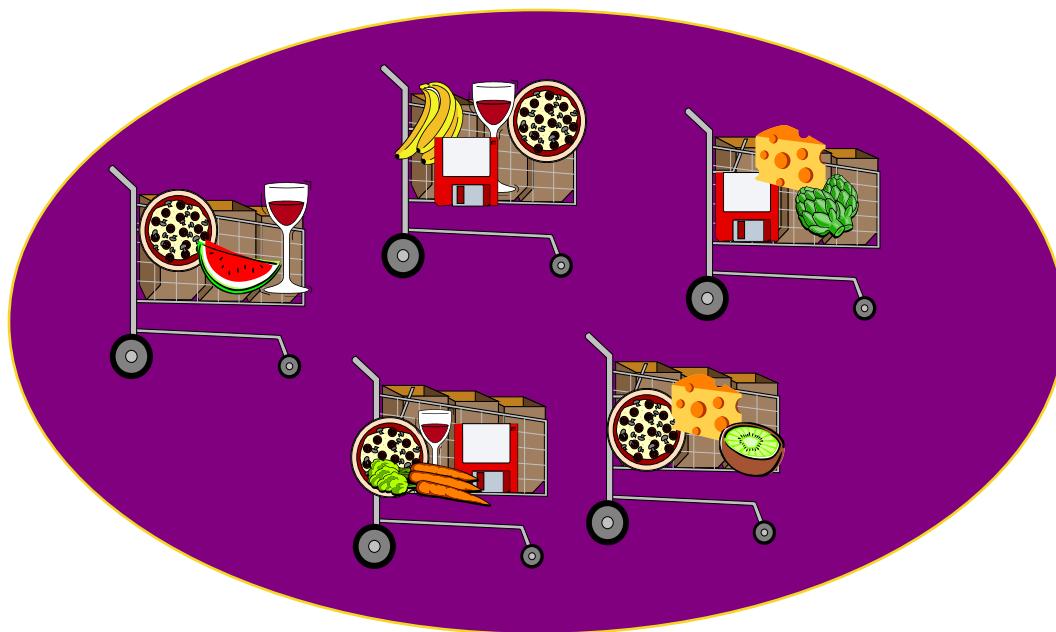
- An application of Machine Learning techniques
 - Data Mining applications exist that examine large pre-existing databases in order to generate new information
 - It solves problems that humans can not solve, because the data involved is too large, noisy ...



Detecting cancer risk molecules is one example.

Data Mining

- A similar application:
 - In marketing products ...

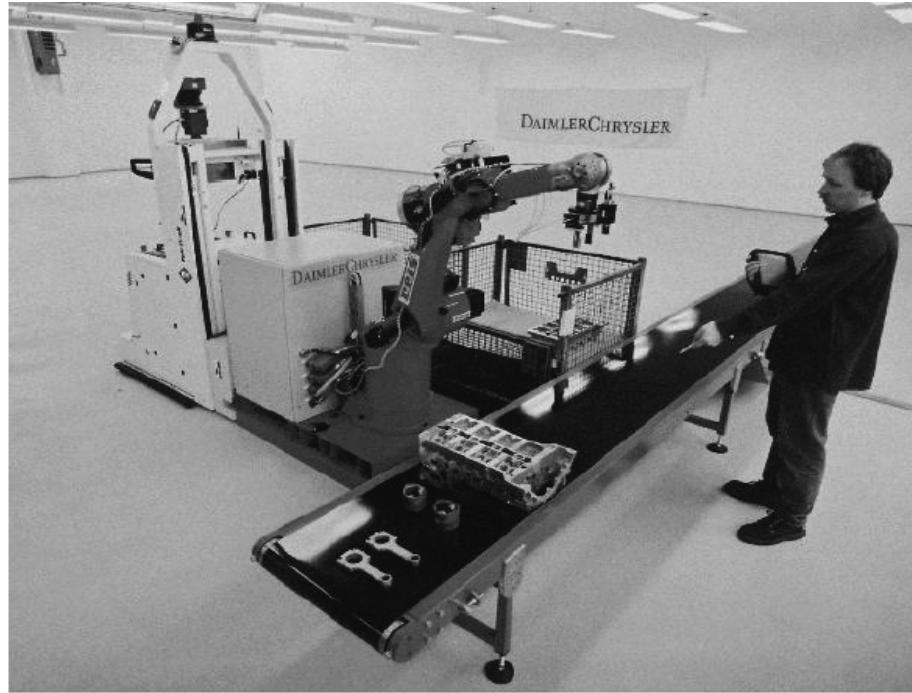


Predicting customer behavior in supermarkets is another.

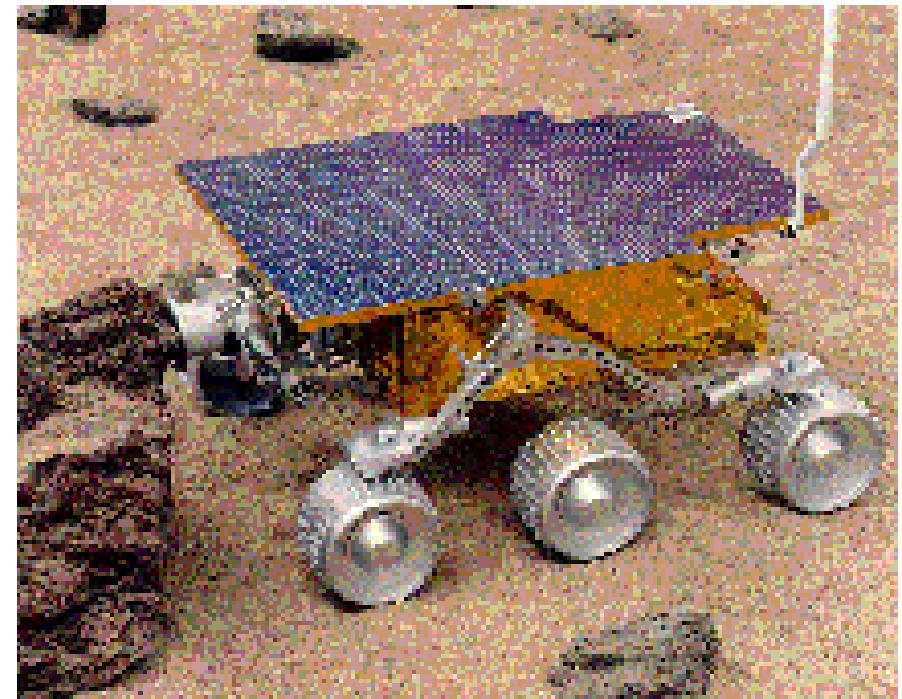
- Fraud Detection
 - 1997, Credit card fraud detection

Robotics

- Traditional Robots



- Mars Rover (1996)
 - Exploring Mars

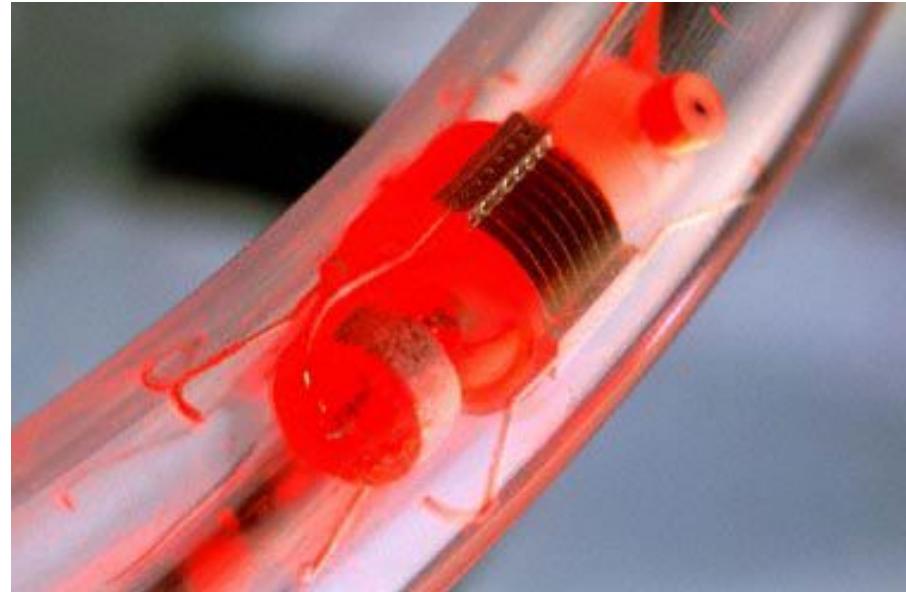


Robotics

- Wire repair



- Pipe inspector



How far is the field advanced?

- Very far
 - Humanoid robots
 - Natural language interfaces
 - Expert systems
 - ...
- Not far at all compared to humans
 - Integration?
 - Learning?

Exercise

- Describe the Turing Test
- According to Alan Turing, When does a computer pass the intelligence / Turing test?
 - For the above two, read Chapters 1 & 26 of *Artificial Intelligence: A Modern Approach* by Russell and Norvig
- Search on the internet about Deep Blue and read on it
- Read Garry Kasparov's interview
 - Find this in the slide presentation: *Deep Blue defeats Kasparov*
- www – Look up the ALICE chatting robot (or ELIZA the therapist) on the internet and chat with 'her'. In your opinion is she intelligent?
- Search on the internet on MYCIN or PROSPECTOR, knowledge-based system and read on it. How is knowledge stored?
- In preparation for the next class, look at Chapter 2: Intelligent Agents on book by Russell and Norvig
- Read: Marvin Minsky, Pioneer in Artificial Intelligence, Dies at 88

<https://www.nytimes.com/2016/01/26/business/marvin-minsky-pioneer-in-artificial-intelligence-dies-at-88.html>

Exercise

- What is Intelligence?
- What is Artificial Intelligence?
- For each of the below sets of statements: Is the latter statement true, and does it imply the former?
 - ▢ “Surely computers cannot be intelligent – They can only do what their programmers tell them”
 - ▢ “Surely animals cannot be intelligent – They can only do what their genes tell them”
- Discuss the Chinese Room Experiment
- Discuss what an expert system is and give two new examples
- Discuss the various disciplines that contributed to the development of AI
- Discuss the events in the development of AI

COMP 308

ARTIFICIAL INTELLIGENCE

PART 2 – INTELLIGENT AGENTS

Njeri Ireri

Jan – April 2020

We Shall Discuss

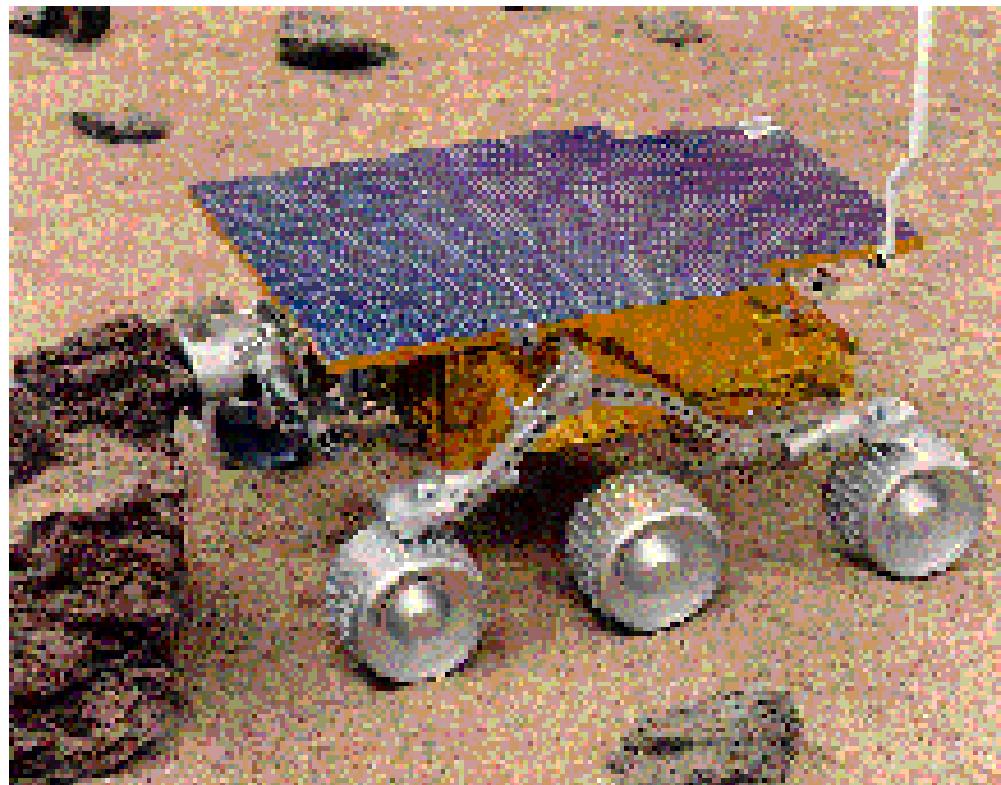
- What an agent is
- What an agents does
- How it is related to its environment
- How it is evaluated
- And how we might go about building one

Recap

- The mission of AI experts is to design / create
 - Computer programs that have some intelligence / can do some intelligent tasks / or can do tasks that require some intelligence
 - Computer programs capable of acting autonomously (independently) exhibiting control over their internal state

Mars Rover

- It is an autonomous device:
 - ▣ Exploring Mars, sending pictures to earth, camera, obstacle sensor, wheels, steering, turning,..

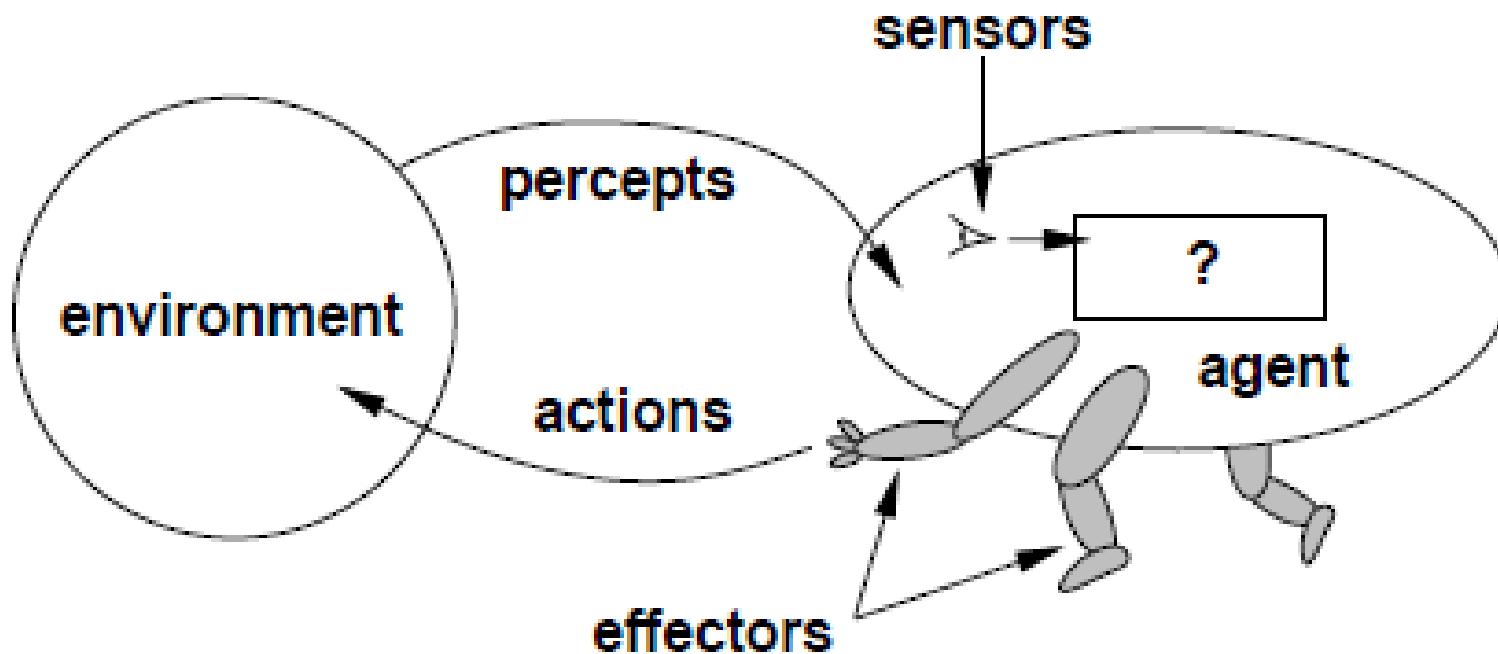


What is an (Intelligent) Agent?

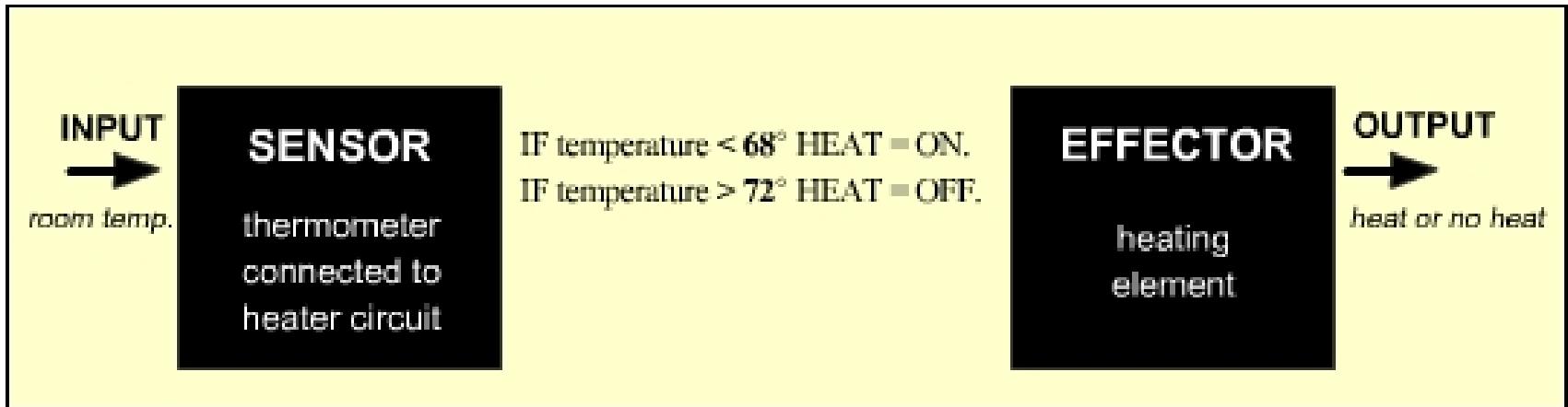
- Agent: An over-used, over-loaded, and misused term
- An **agent** is anything that can be **viewed** as **perceiving** its **environment** through **sensors** and **acting** upon that environment through its **effectors** to maximize progress towards its **goals**
- A **human agent** has eyes, ears, and other organs for sensors, and hands, legs, mouth, and other body parts for effectors
- A **robotic agent** substitutes cameras and infrared range finders for the sensors and various motors for the effectors
- **PAGE** (**P**ercepts, **A**ctions, **G**oals, **E**nvironment)
- An agent is task-specific & specialized, i.e.. It has well-defined goals and environment

Intelligent Agents

- Agents interact with environments through sensors and effectors



Intelligent Agents

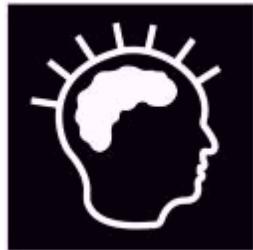


Thermostat Agent

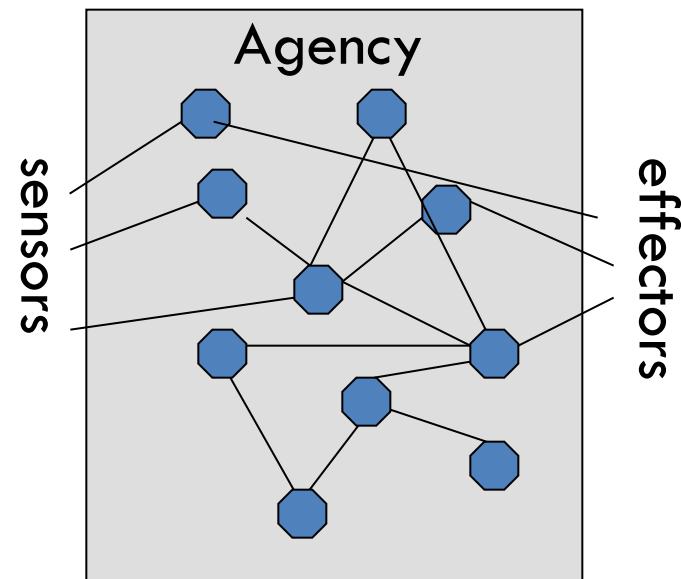
- The is a simple agent that responds to a very specific feature of the environment with only three possible actions: turn heat on or turn heat off or take no action
- Such an agent does not really qualify as intelligent or autonomous, because its behaviour is too limited and lacks flexibility and adaptability to qualify as autonomous by definition
- Notice that even with a simple thermostat the environment determines the action of the agent and the agent's action, in turn, modifies the environment, in a relationship of mutual determination. When the temperature falls to 68 F the heat is turned on, but this brings the ambient temperature of the room to 80 F, which triggers an action to turn the heater off
- The environment is generally the domain or world of the agent

Intelligent Agents and Artificial Intelligence

- Human mind as network of thousands or millions of agents all working in parallel. To produce real artificial intelligence, this school holds, we should build computer systems that also contain many agents and systems for arbitrating among the agents' competing results



- Distributed decision-making and control
- Challenges:
 - Action selection: What next action to choose
 - Conflict resolution



Agent Research Areas

- We can split agent research into two main strands:
 - Distributed Artificial Intelligence (DAI) – Multi-Agent Systems (MAS) (1980 – 1990)
 - Much broader notion of "agent" (1990's – present)
 - interface, reactive, mobile, information

Examples of Intelligent Agents – Towards Autonomous Vehicles



A Windscreen Wiper Agent

- How do we design a agent that can wipe the windscreens when needed?
 - Goals?
 - Percepts ?
 - Sensors?
 - Effectors / Actuators ?
 - Actions ?
 - Environment ?

A Windscreen Wiper Agent

- Goals: To keep windscreens clean and maintain good visibility
- Percepts: Raining, Dirty, Clear
- Sensors: Camera (moisture sensor)
- Effectors: Wipers (left, right, back)
- Actions: Off, Slow, Medium, Fast
- Environment: Nairobi city, pot-holed roads, highways, weather ... Nyahururu town?

Interacting Agents

Collision Avoidance Agent (CAA)

- Goals: Avoid running into obstacles
- Percepts ?
- Sensors?
- Effectors ?
- Actions ?
- Environment: River Road, Nairobi

Lane Keeping Agent (LKA)

- Goals: Stay in current lane
- Percepts ?
- Sensors?
- Effectors ?
- Actions ?
- Environment: Highway

Interacting Agents

Collision Avoidance Agent (CAA)

- Goals: Avoid running into obstacles
- Percepts: Obstacle distance, velocity, trajectory
- Sensors: Vision, proximity sensing
- Effectors: Steering Wheel, Accelerator, Brakes, Horn, Headlights
- Actions: Steer, speed up, brake, blow horn, signal (headlights)
- Environment: River Road, Nairobi

Lane Keeping Agent (LKA)

- Goals: Stay in current lane
- Percepts: Lane center, lane boundaries
- Sensors: Vision
- Effectors: Steering Wheel, Accelerator, Brakes
- Actions: Steer, speed up, brake
- Environment: Highway

Examples of Intelligent Agents

Agent	Goal	Percepts	Sensors	Effectors	Actions	Environment
Bin-Picking Robot	Parts in correct bins	Images	Vision	robotic arm	Grasp objects; Sort into bins	Parts in correct bins
Medical Diagnosis	Healthy patient	Patient symptoms, tests	Vision, Keyboard entry of symptoms,		Tests and treatments	Patient & hospital
Webcrawler Softbot	Collect information on a subject	Web pages			Follow links, pattern matching	Internet
Financial forecasting Software	Pick stocks to buy & sell	Financial data			Gather data on companies	Stock market, company reports

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	Correct image categorization	Downlink from orbiting satellite	Display of scene categorization	Color pixel arrays
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Purity, yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, suggestions, corrections	Keyboard entry

Examples of Intelligent Agents

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard

Examples of Intelligent Agents

Agent	Goal	Percepts	Sensors	Effectors	Actions	Environment
Smartphone Personal Assistant						
AI bot in a gaming platform						
Online Chat bot						

The Right Thing = The Rational Action

- An **agent** gets percepts one at a time, and maps this percept sequence to actions, it should **act rationally** upon its environment in order to meet its design objectives
- A **rational agent** always performs **right action**
- **Rationality** is the status of being reasonable, sensible, and having good sense of judgment
- **Rational Action:** The action that maximizes the expected value of the performance measure given the percept sequence to date
 - Rational = Best ?
 - Rational = Optimal ?
 - Rational = Omniscience ?
 - Rational = Successful ?

The Right Thing = The Rational Action

- **Rational Action:** The action that maximizes the expected value of the performance measure given the percept sequence to date
 - Rational = Best Yes, to the best of its knowledge
 - Rational = Optimal Yes, to the best of its abilities (incl. its constraints)
 - Rational \neq Omiscience
 - Rational \neq Successful
- That leaves us with the problem of deciding **how** and **when** to evaluate the agent's performance

Behavior and Performance of IAs

- **How to Evaluate an Agent's Behavior/Performance?**
 - To measure **Rationality of an IA** consider the following factors:
 - The **performance measures**, which determine the degree of success
 - Agent's **Percept Sequence** (history of all that an agent has perceived till date) till now
 - The agent's **prior knowledge about the environment**
 - The **actions** that the agent can carry out

Behavior and Performance of IAs

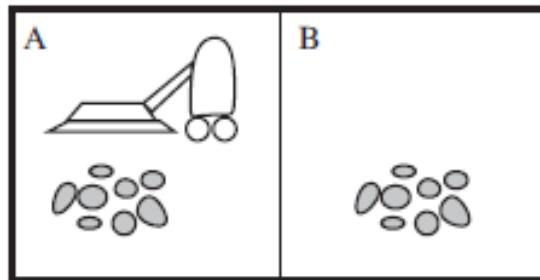
- **Agent Function** = Perception (sequence) to Action Mapping

$$f : \mathcal{P}^* \rightarrow \mathcal{A}$$

- **Ideal mapping:** specifies which actions an agent ought to take at any point in time
- **Description:** Look-Up-Table,...
- **Performance measure:** a *subjective* measure to characterize how successful an agent is (e.g., speed, power usage, accuracy, money, etc.)
- (degree of) **Autonomy:** to what extent is the agent able to make decisions and actions on its own?

Behavior and Performance of IAs

Example: Vacuum-cleaner Agent whose world with just two locations



- Percepts: Location and contents, e.g. [A, Dirty]
- Actions: Left, Right, Suck, NoOp
- Agent's Function → look-up table
 - ▣ For many agents this is a very large table

Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
:	:

How is an Agent different from other software?

- Agents are **autonomous**, that is they act on behalf of the user
- Agents contain some level of **intelligence**, from fixed rules to learning engines that allow them to adapt to changes in the environment
- Agents don't only act **reactively**, but sometimes also **proactively**
- Agents have **social ability**, that is they communicate with the user, the system, and other agents as required
- Agents may also **cooperate** with other agents to carry out more complex tasks than they themselves can handle
- Agents may **migrate** from one system to another to access remote resources or even to meet other agents

Environment Types

- **Characteristics**
 - Accessible (observable) vs. inaccessible
 - An environment is accessible if the sensors detect all aspects that are relevant to the choice of action
 - Deterministic vs. non-deterministic
 - If the next state of the environment is completely determined by the current state and the actions selected by the agents, then we say the environment is deterministic
 - Episodic vs. non-episodic
 - In an episodic environment, the agent's experience is divided into "episodes." Each episode consists of the agent perceiving and then acting

Environment Types

- **Characteristics**
 - Static vs. dynamic
 - If the environment can change while an agent is deliberating, then we say the environment is dynamic for that agent; otherwise it is static
 - Discrete vs. continuous
 - If there are a limited number of distinct, clearly defined percepts and actions we say that the environment is discrete
 - Hostile vs. friendly
 - This depends on the agent perception

Environment Types

Environment	Accessible	Deterministic	Episodic	Static	Discrete
Operating System					
Virtual Reality					
Office Environment					
Mars					

Environment Types

Environment	Accessible	Deterministic	Episodic	Static	Discrete
Operating System	Yes	Yes	No	No	Yes
Virtual Reality	Yes	Yes	Yes/No	No	Yes/No
Office Environment	No	No	No	No	No
Mars	No	Semi	No	Semi	No

- The environment types largely determine the agent design

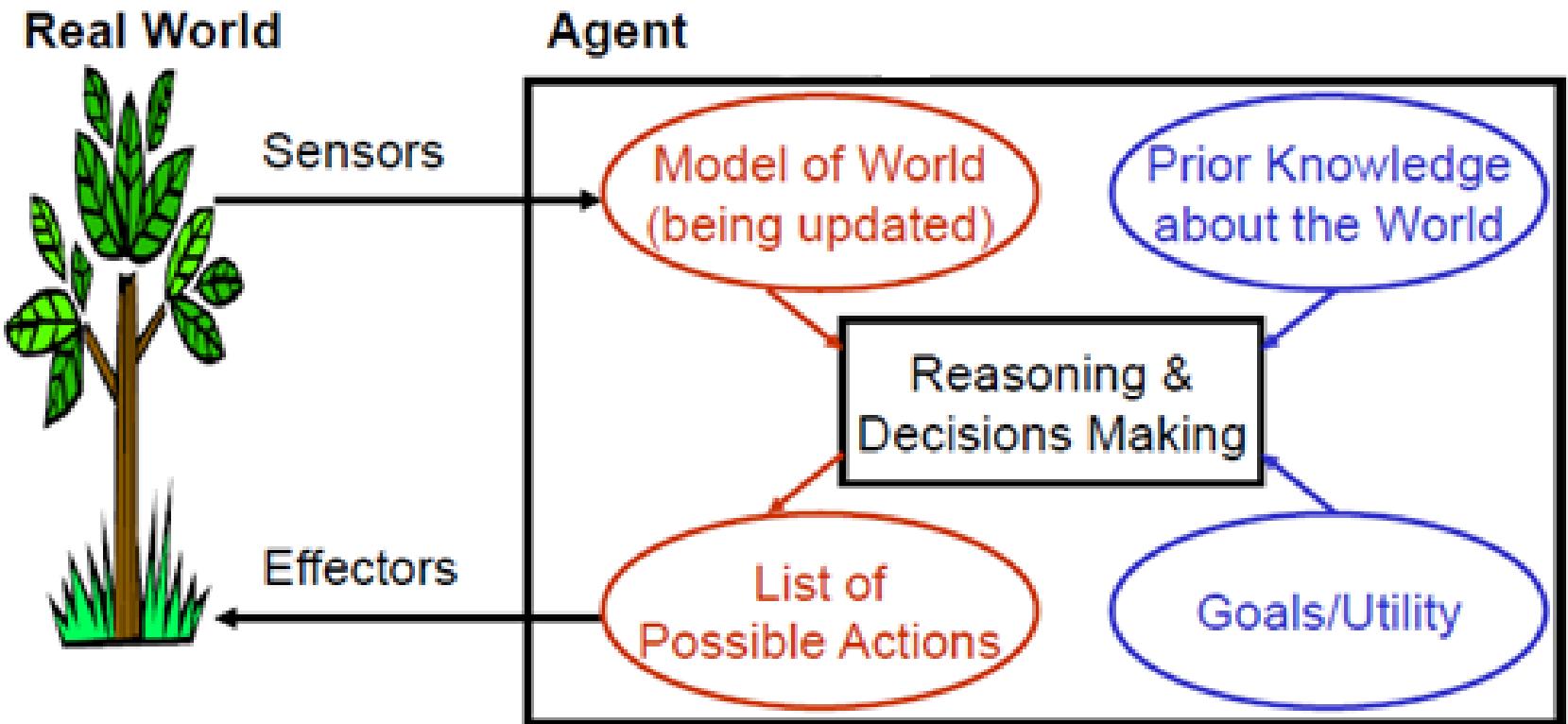
Environment Types

Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle	Fully	Single	Deterministic	Sequential	Static	Discrete
Chess with a clock	Fully	Multi	Deterministic	Sequential	Semi	Discrete
Poker	Partially	Multi	Stochastic	Sequential	Static	Discrete
Backgammon	Fully	Multi	Stochastic	Sequential	Static	Discrete
Taxi driving	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous
Medical diagnosis	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Image analysis	Fully	Single	Deterministic	Episodic	Semi	Continuous
Part-picking robot	Partially	Single	Stochastic	Episodic	Dynamic	Continuous
Refinery controller	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Interactive English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

Structure of Intelligent Agents

- **Agent** = architecture + program
- **Agent program:** the implementation of $f : \mathcal{P}^* \rightarrow \mathcal{A}$, the agent's perception-action mapping
 - function:** Skeleton-Agent(*Percept*) **returns** *Action*
 memory \leftarrow UpdateMemory(memory, *Percept*)
 Action \leftarrow ChooseBestAction(memory)
 memory \leftarrow UpdateMemory(memory, *Action*)
 - return** *Action*
- **Architecture:** a device that can execute the agent program (e.g., general-purpose computer, specialized device, specialized hardware/software, beobot, etc.)

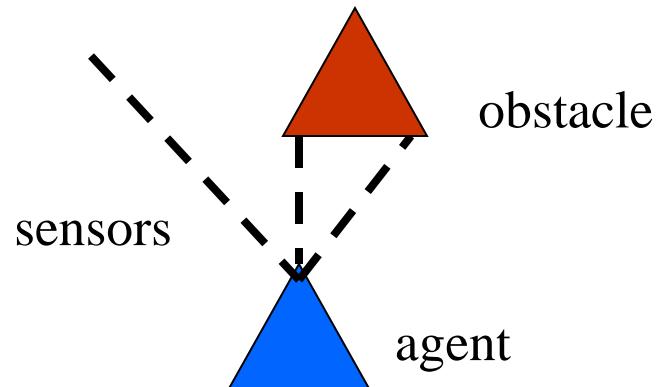
Structure of Intelligent Agents



Using a look-up-table to encode

$$f : \mathcal{P}^* \rightarrow \mathcal{A}$$

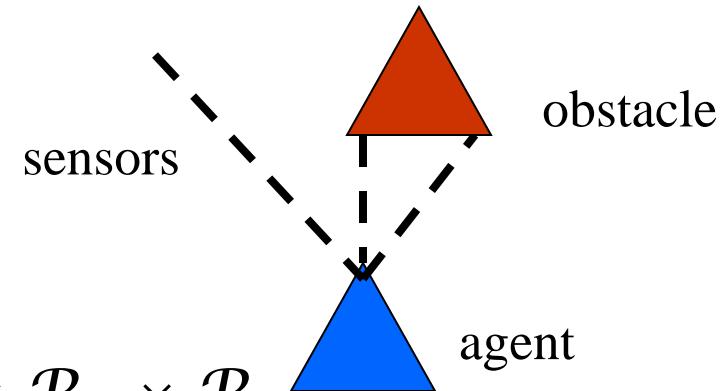
- Example: Collision Avoidance
 - Sensors: 3 proximity sensors
 - Effectors: Steering wheel, Brakes
- How to generate?
- How large?
- How to select action?



Using a look-up-table to encode

$$f : \mathcal{P}^* \rightarrow \mathcal{A}$$

- **Example:** Collision Avoidance
 - Sensors: 3 proximity sensors
 - Effectors: Steering wheel, Brakes
- **How to generate:** for each $p \in \mathcal{P}_l \times \mathcal{P}_m \times \mathcal{P}_r$ generate an appropriate action, $a \in S \times \mathcal{B}$



- **How large:** size of table = #possible percepts times # possible actions = $|\mathcal{P}_l| |\mathcal{P}_m| |\mathcal{P}_r| |S| |\mathcal{B}|$
E.g., $P = \{\text{close, medium, far}\}^3$
 $A = \{\text{left, straight, right}\} \times \{\text{on, off}\}$
then size of table = $27 * 3 * 2 = 162$

- **How to select action?** Search

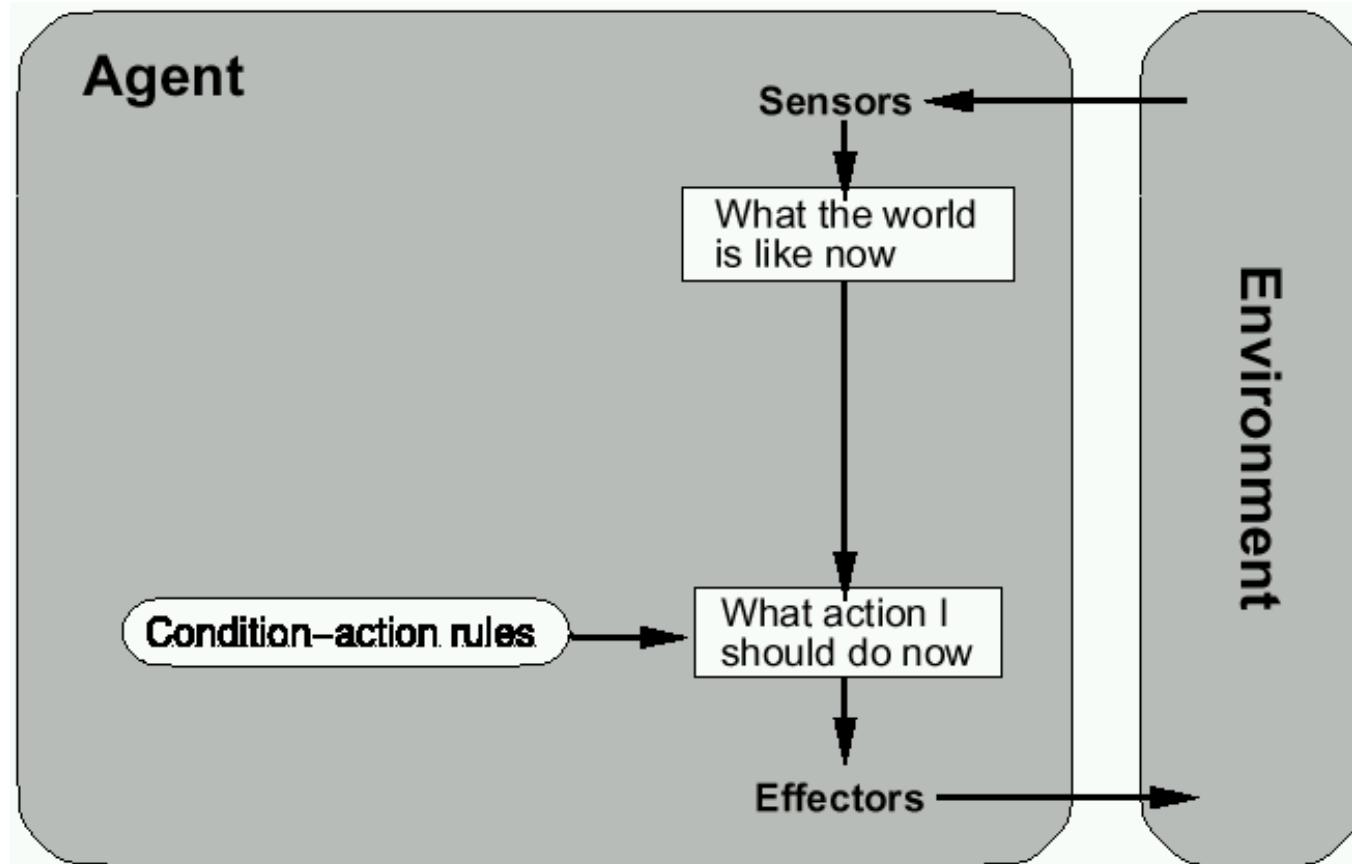
Exercise

- Consider any of the agent examples discussed earlier. The aim of this exercise is to determine the ideal mapping for the agent. Determine:
 - How to generate a look up table for the agent?
 - How large the agents look up table would be?

Agent Types

- Reflex agents
- Reflex agents with internal states
- Goal-based agents
- Utility-based agents
- Learning agents

Reflex agents

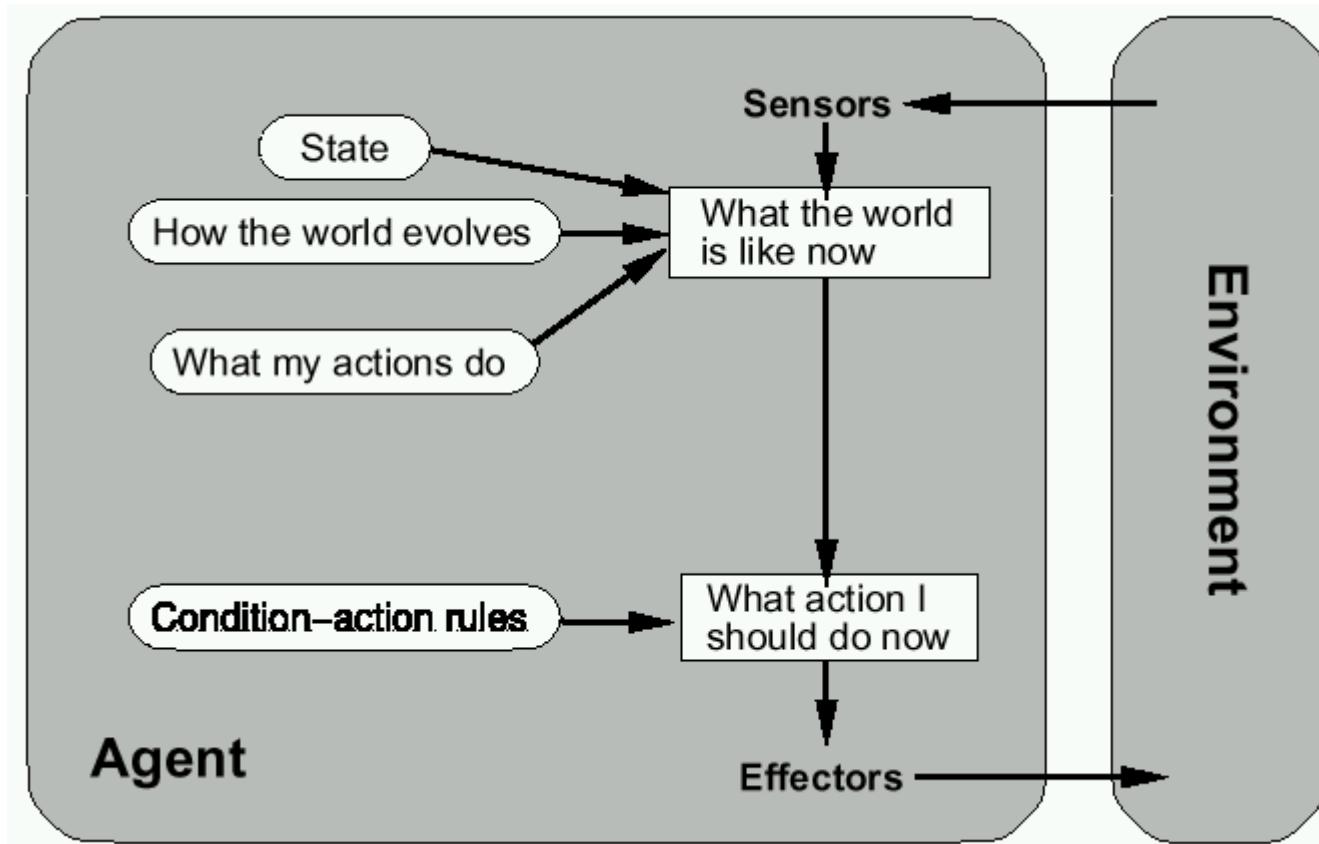


Reflex agents

- Reflex/Reactive agents do not have internal symbolic models
i.e. **do not have memory of past** world states or percepts
- So, actions depend solely on **current percept**. They act by stimulus-response to the current state of the environment
- Action becomes a "reflex"
- Each reactive agent is simple and interacts with others in a basic way
- Complex patterns of behavior emerge from their interaction

- **Benefits:** robustness, fast response time
- **Challenges:** scalability, how intelligent?
and how do you debug them?

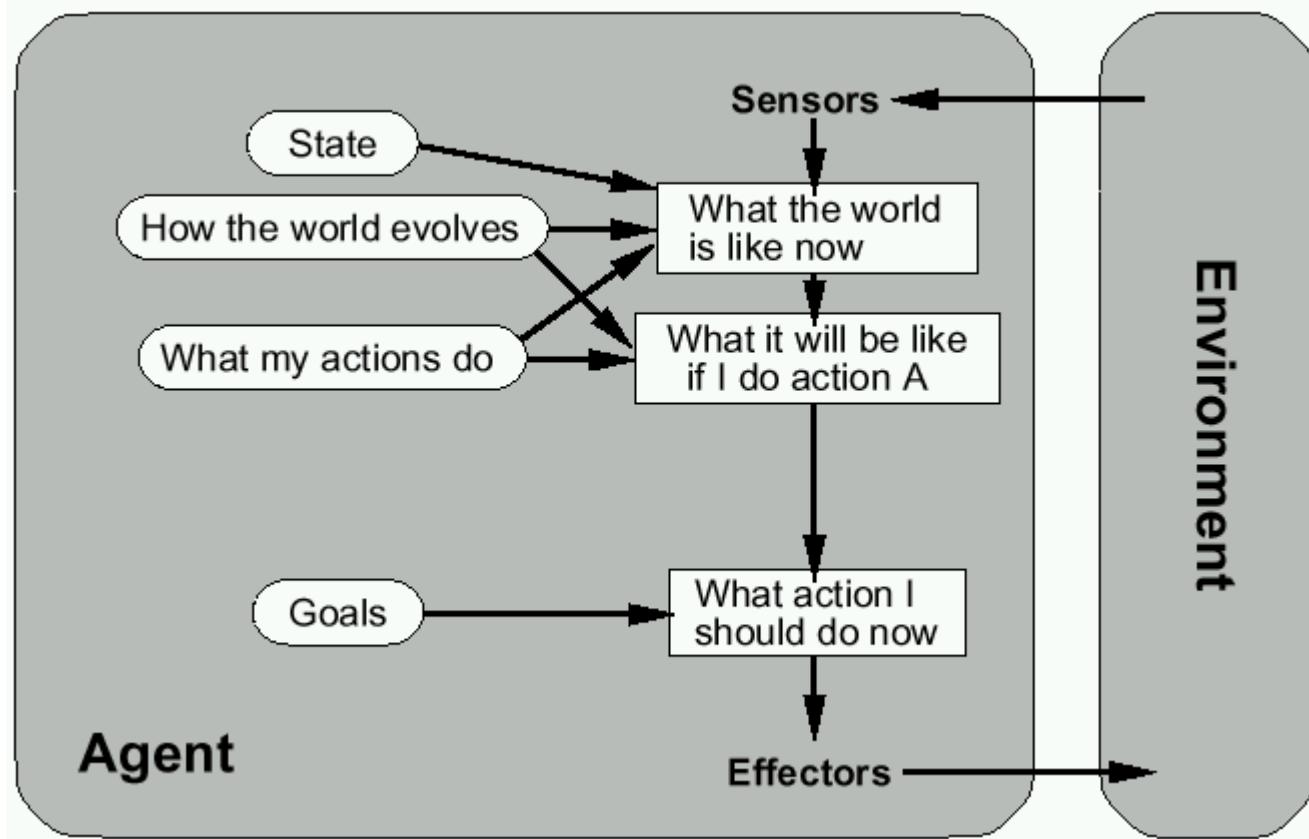
Reflex agents with state



Reflex agents with state

- Also known as Model-based reflex agents
- Key difference with regards to simple reflex agents:
 - Agents have **internal state**, which is used to keep track of past states of the world
 - Agents have the ability **to represent change in the World**
- **Main idea:** build complex, intelligent robots by decomposing behaviors into a hierarchy of skills, each defining a percept-action cycle for one very specific task
- **Examples:** collision avoidance, wandering, exploring, recognizing doorways, etc.

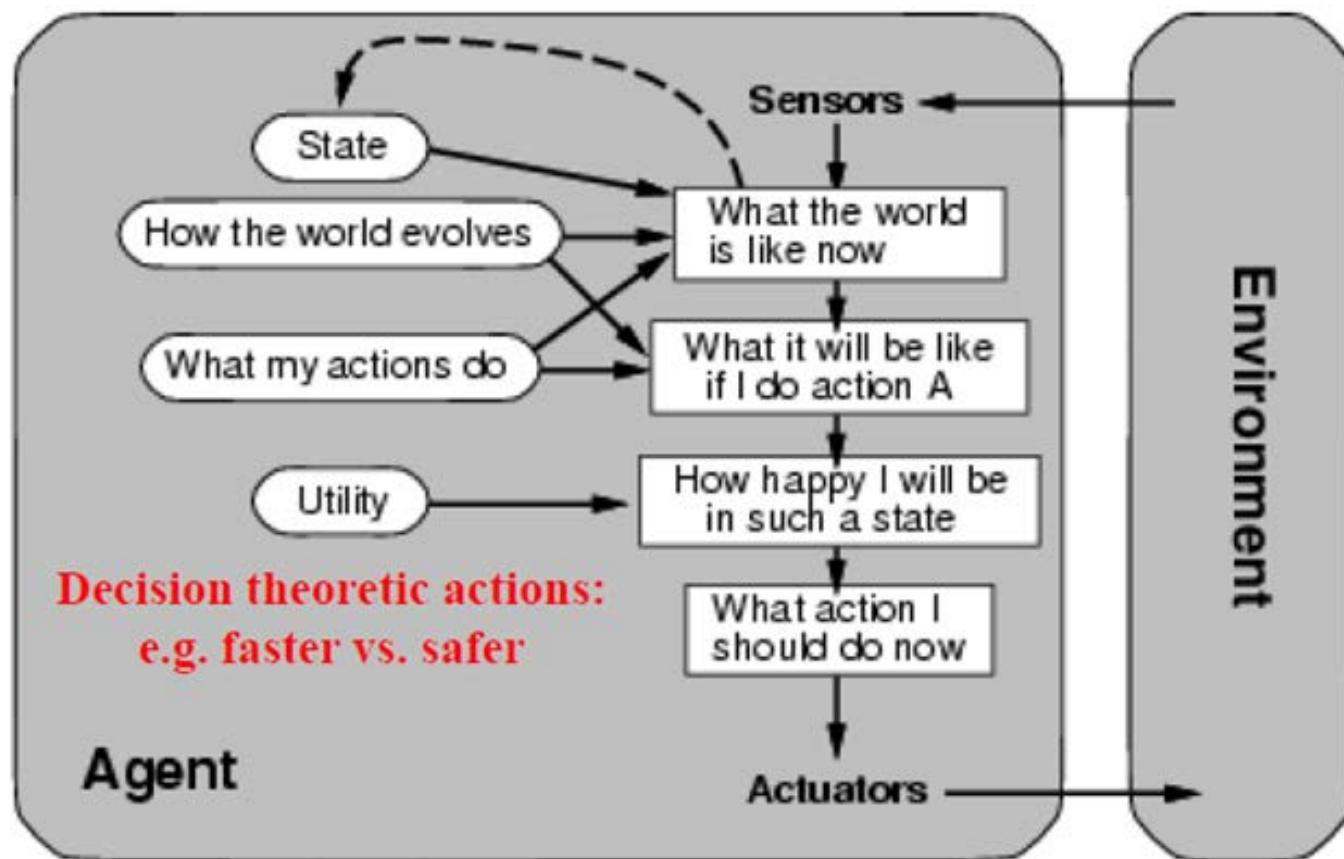
Goal-based agents



Goal-based agents

- Key difference with regards to Model-Based Agents:
 - In addition to state information, have **goal information** that **describes desirable situations to be achieved**
- Agents of this kind take future events into consideration.
 - **What sequence of actions can I take to achieve certain goals?**
- Choose actions so as to (eventually) achieve a (given or computed) goal
- ***problem solving and search!***

Utility-based agents



Utility-based agents

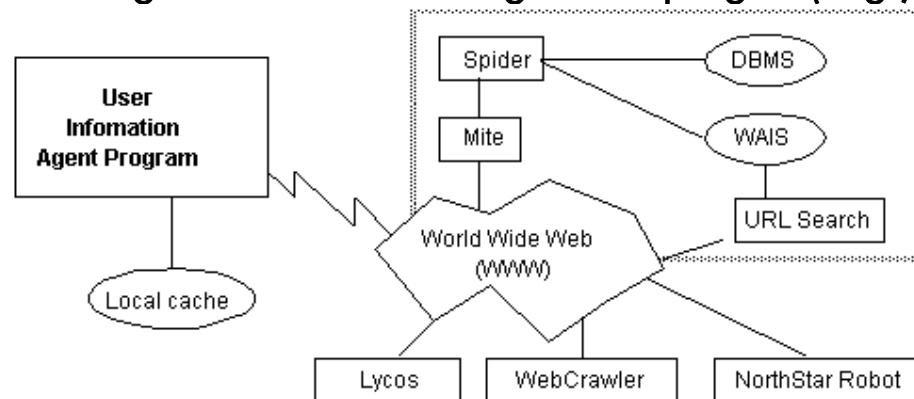
- When there are **multiple possible alternatives**, how to decide which one is best?
- Goals are qualitative: A goal specifies a crude distinction between a happy and unhappy state, but often a more general performance measure that describes "degree of happiness"
 - **Utility function U : State $\rightarrow \mathbb{R}$** indicating a measure of success or happiness when at a given state
- Important for making tradeoffs: Allows decisions comparing choice between conflicting goals, and choice between likelihood of success and importance of goal (if achievement is uncertain)
- **Use decision theoretic models: e.g. faster vs. safer**

Mobile agents

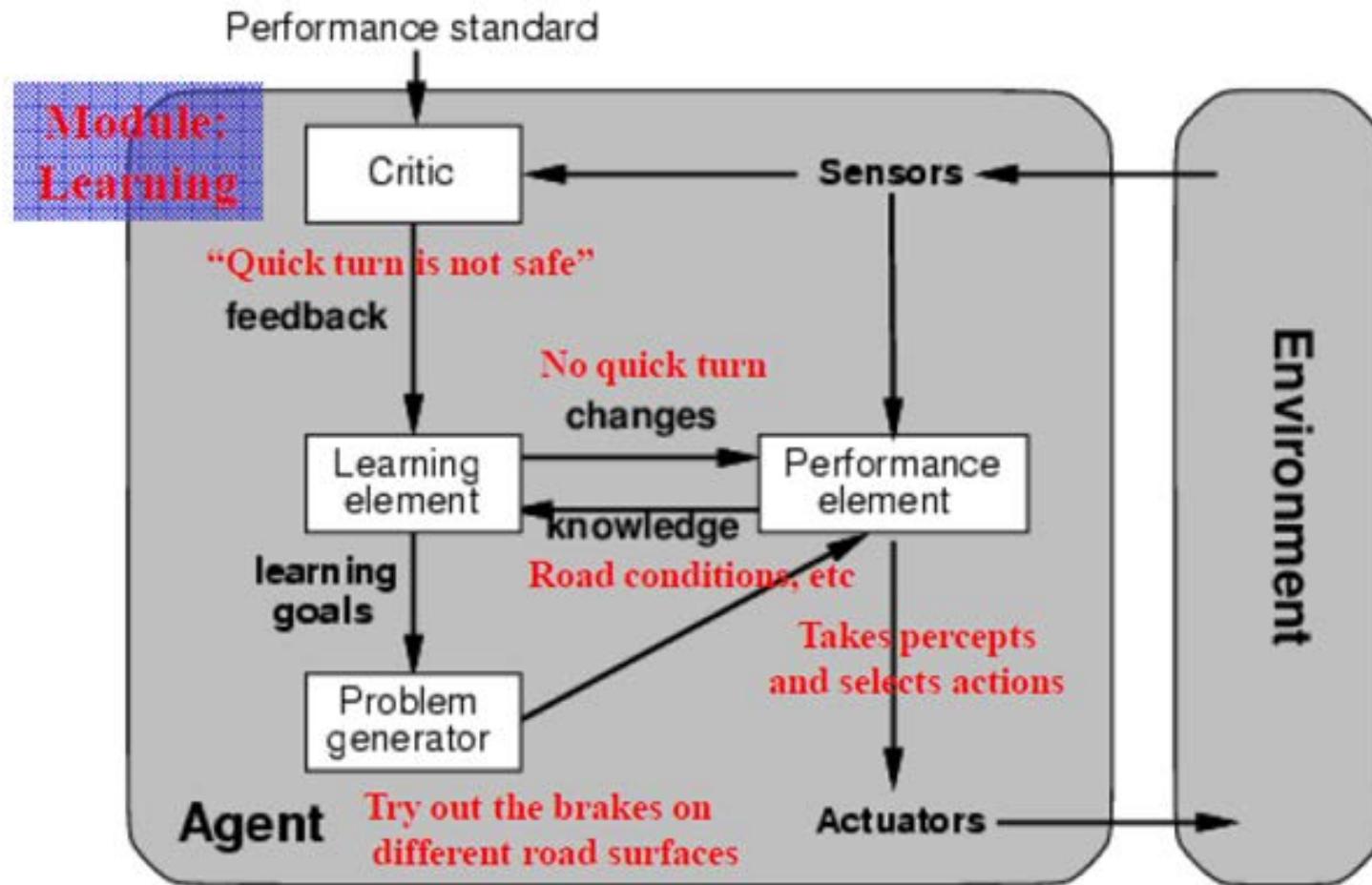
- Programs that can migrate from one machine to another
- Execute in a platform-independent execution environment
- Require agent execution environment (places)
- Mobility not necessary or sufficient condition for agenthood
- Practical but non-functional advantages:
 - Reduced communication cost (e.g., from PDA)
 - Asynchronous computing (when you are not connected)
- Two types:
 - One-hop mobile agents (migrate to one other place)
 - Multi-hop mobile agents (roam the network from place to place)
- Applications:
 - Distributed information retrieval.
 - Telecommunication network routing.

Information agents

- Manage the explosive growth of information.
- Manipulate or collate information from many distributed sources.
- Information agents can be mobile or static.
- Examples:
 - [BargainFinder](#) comparison shops among Internet stores for CDs
 - [FIDO](#) the Shopping Doggie (out of service)
 - [Internet Softbot](#) infers which internet facilities (finger, ftp, gopher) to use and when from high-level search requests.
- Challenge: ontologies for annotating Web pages (e.g., SHOE).



Learning agents



Learning agents

- Learning Agents **Adapt** and **Improve** over time
- More complicated when agent needs to learn **utility information:**
 - Reinforcement learning (based on action payoff)

Summary

- **Intelligent Agents:**
 - Anything that can be **viewed** as **perceiving** its **environment** through **sensors** and **acting** upon that environment through its **effectors** to maximize progress towards its **goals**.
 - PAGE (Percepts, Actions, Goals, Environment)
 - Described as a Perception (sequence) to Action Mapping:
$$f : \mathcal{P}^* \rightarrow \mathcal{A}$$
 - Using look-up-table, closed form, etc.
- **Agent Types:** Reflex, state-based, goal-based, utility-based, learning
- **Rational Action:** The action that maximizes the expected value of the performance measure given the percept sequence to date

Summary

- Simple reflex agents – are based on condition-action rules, implemented with an appropriate production system. They are stateless devices which do not have memory of past world states
- Agents with memory - Model-based reflex agents – have internal state, which is used to keep track of past states of the world.
- Agents with goals – Goal-based agents – are agents that, in addition to state information, have goal information that describes desirable situations. Agents of this kind take future events into consideration
- Utility-based agents – base their decisions on classic axiomatic utility theory in order to act rationally
- Learning agents – they have the ability to improve performance through learning

Exercise

- Differentiate between:
 - Agent
 - Rational Agent
 - Autonomous Agent

ARTIFICIAL INTELLIGENCE

PART 3 – PROBLEM SOLVING - SEARCHING

Njeri Ireri

July – October 2021

We Shall Discuss

- Introduction to problem solving
- Problem Solving Techniques
- Search as a problem solving technique
- Problem Definition
- Search Terminology
- Evaluating a search

Workers are always Searching



Brief info on Ants, <http://www.winnipeg-bugline.com/ants.html>,
<http://www.boston.com/globe/magazine/2002/0623/frontiers.htm>

Problem Solving Techniques in A.I

- Broad Approaches
 - using search techniques – uninformed, informed...
 - e.g. in Games
 - modeling
 - using Knowledge Base Systems (KBS)
 - using Machine Learning techniques e.g. Artificial Neural Networks, Decision Trees, Case-base reasoning, Genetic algorithms, ..

Searching as a Problem Solving Technique

- **Searching** is the process of looking for the solution of a problem through a set of possibilities (state space)
- **Search conditions** include:
 - Current state – where one is;
 - Goal state – the solution reached; check whether it has been reached;
 - Cost of obtaining the solution
- The **solution** is a path from the current state to the goal state

Searching as a Problem Solving Technique

Process of Searching

- Searching proceeds as follows:
 - Check the current state;
 - Execute allowable actions to move to the next state;
 - Check if the new state is the solution state; if it is not, then the new state becomes the current state and the process is repeated until a solution is found or the state space is exhausted

Search Problem

- The **search problem** consists of finding a **solution plan**, which is a path from the current state to the goal state
- **Representing search problems**
 - A search problem is represented using a directed graph (tree)
 - The states are represented as nodes while the allowed steps or actions are represented as arcs (branches)
- **A search problem is defined by specifying:**
 - State space;
 - Start node;
 - Goal condition, and a test to check whether the goal condition is met;
 - Rules giving how to change states
 - Path cost

Problem Definition - Example, 8 puzzle

5	4	
6	1	8
7	3	2

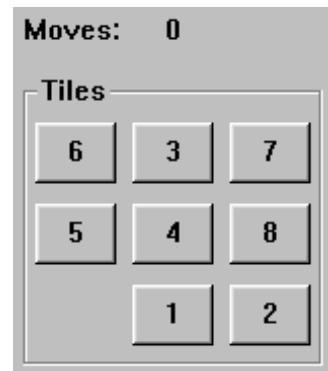
Initial State

1	4	7
2	5	8
3	6	

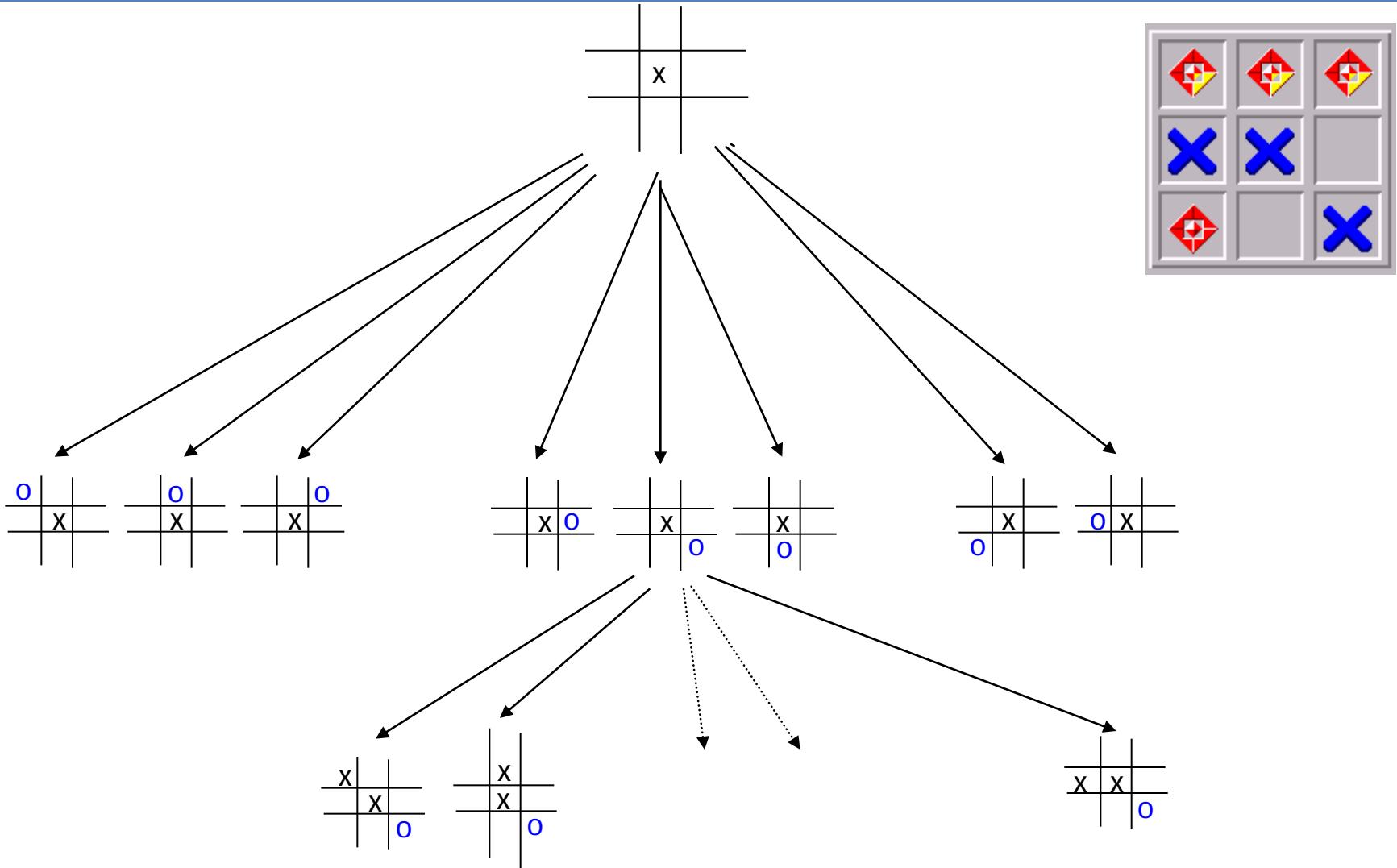
Goal State

Problem Definition - Example, 8 puzzle

- States
 - ▣ A description of each of the eight tiles in each location that it can occupy. It is also useful to include the blank
- Operators/Action
 - ▣ The blank moves left, right, up or down
- Goal Test
 - ▣ The current state matches a certain state (e.g. one of the ones shown on previous slide)
- Path Cost
 - ▣ Each move of the blank costs 1



Problem Definition - Example, tic-tac-toe



Exercise

4. (a) Playing the 8 Puzzle game, draw a search tree to level three for the initial game state given in figure 1
(b) How many moves would you require to complete the game given in figure 1

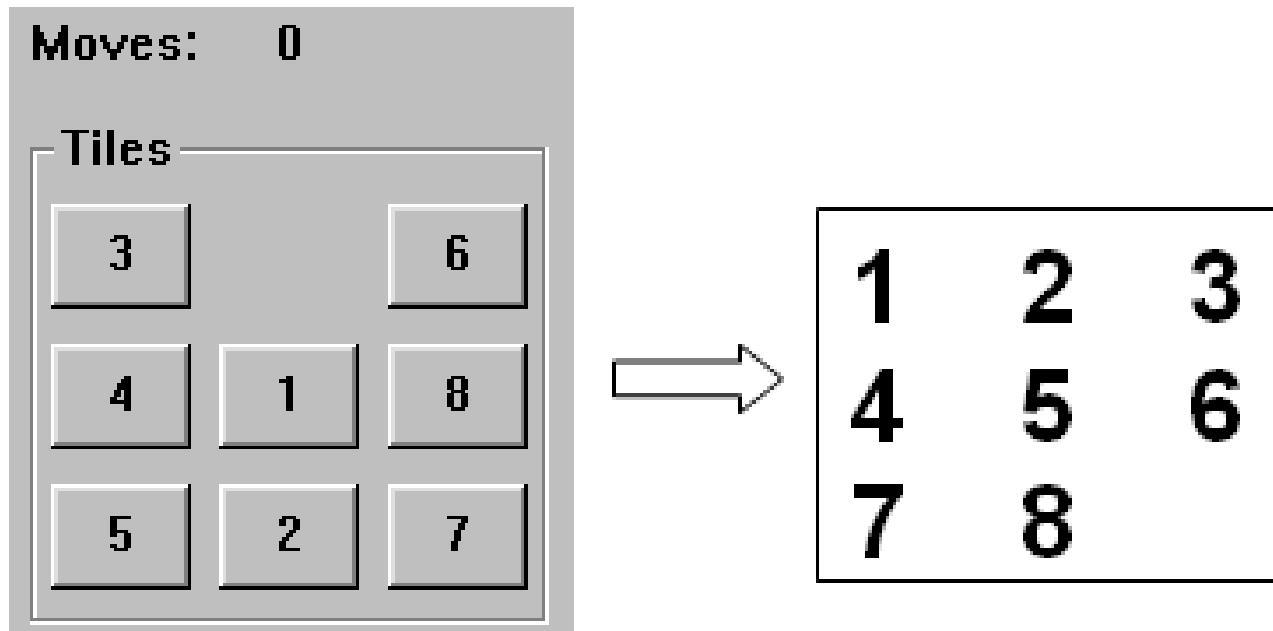
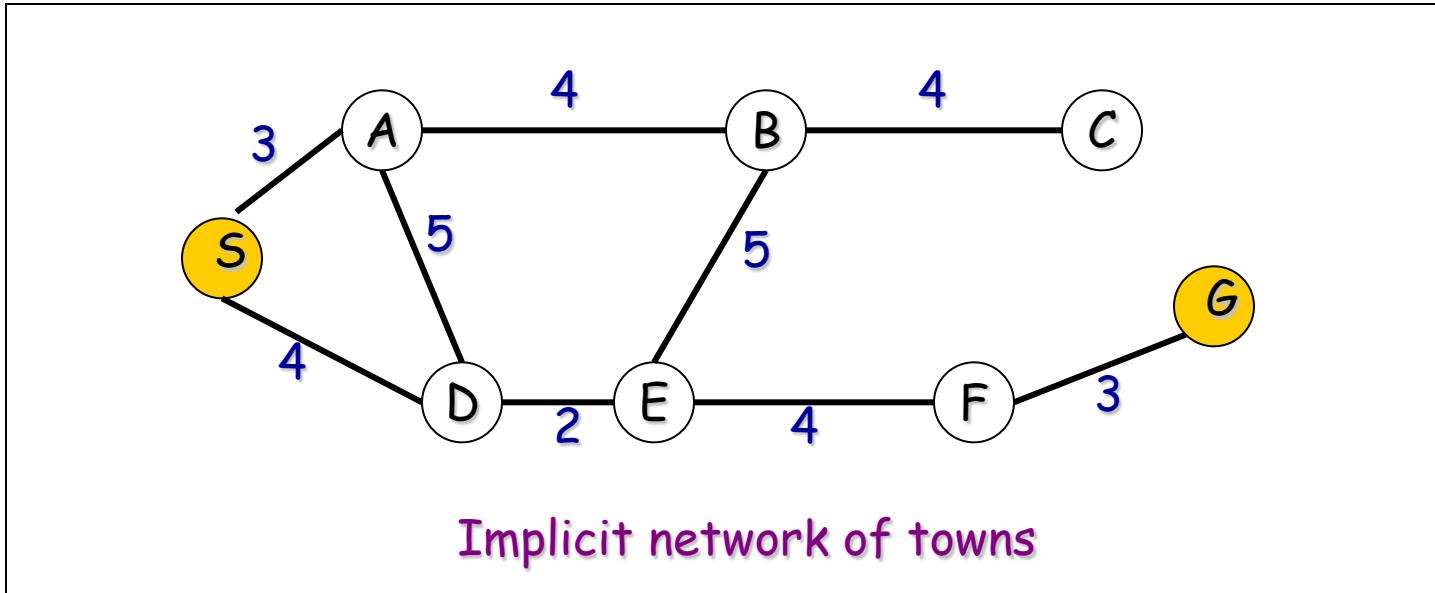


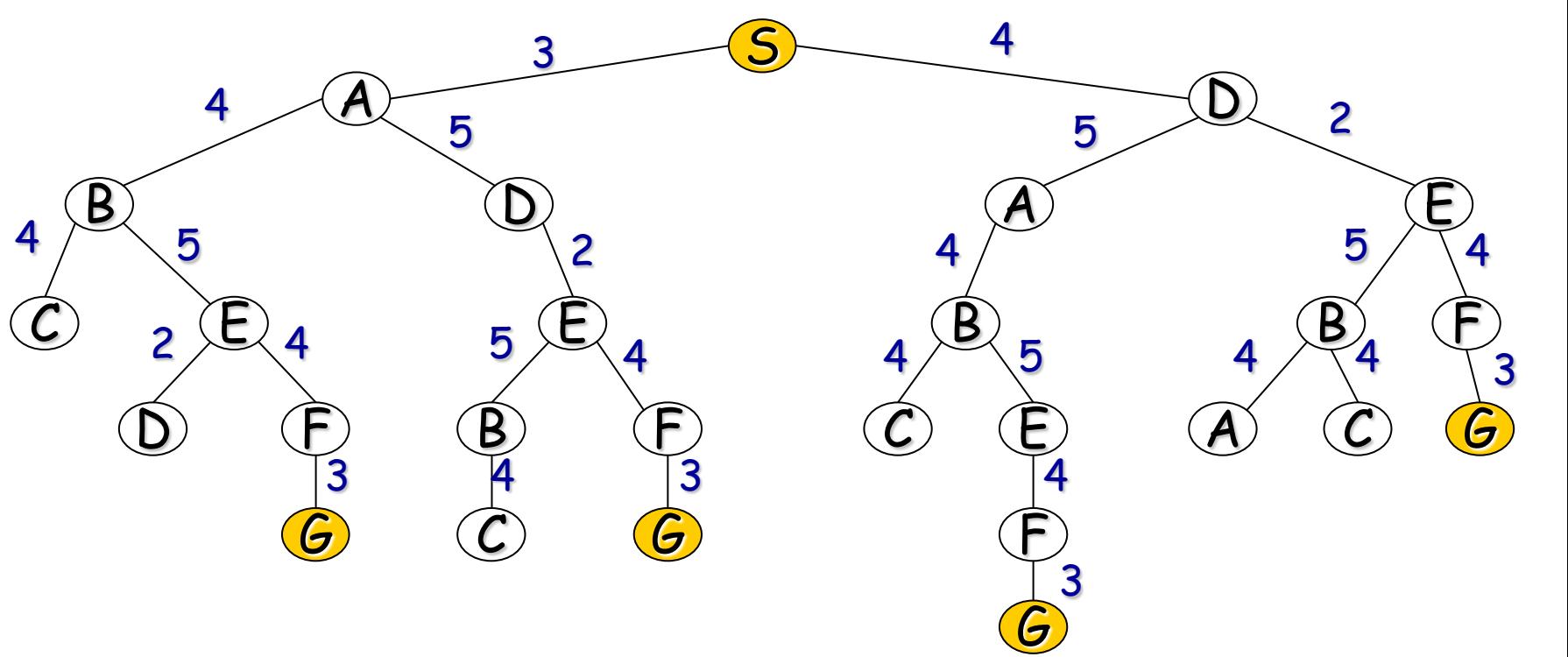
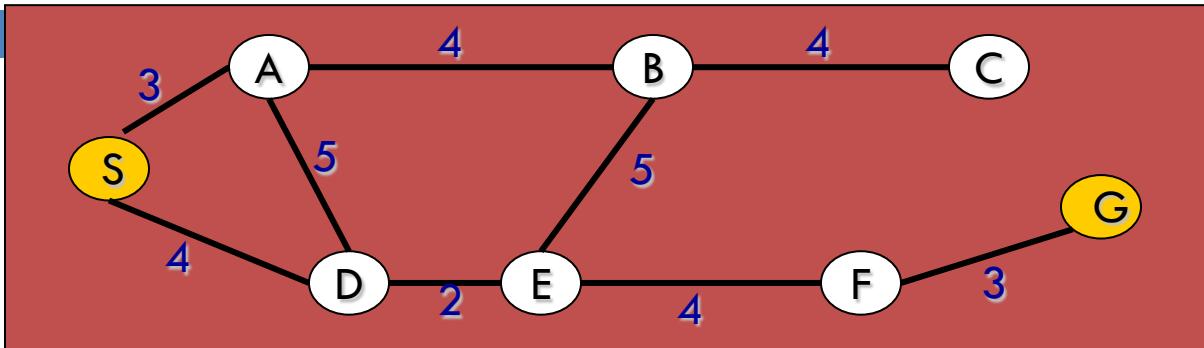
Figure 1. 8 Puzzle (Left-Start State and Right-Goal State)

Tree/Path Example:



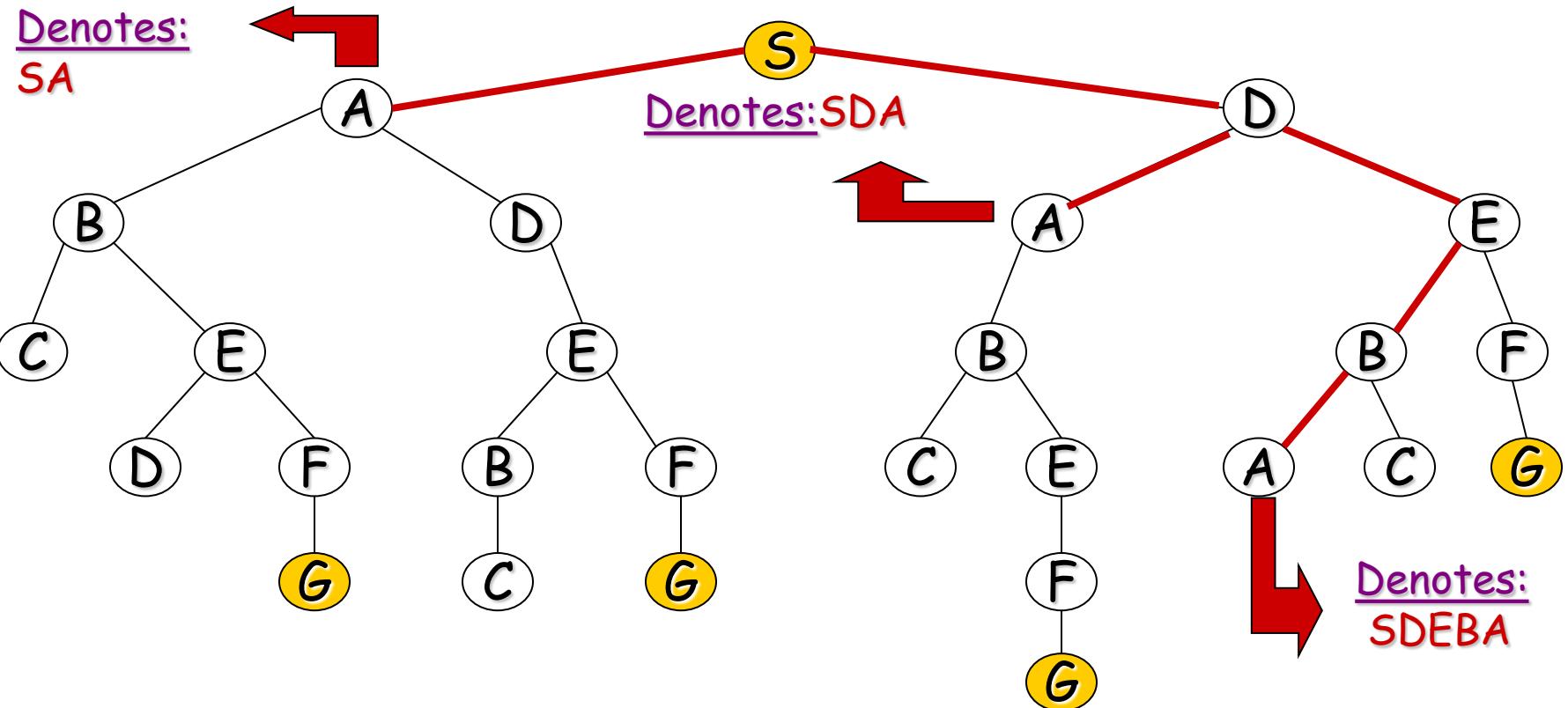
- Two possible tasks:
 - 1. FIND a (the) path. = computational cost
 - 2. TRAVERSE the path. = travel cost
- 2. relates to finding optimal paths

The associated loop-free tree of partial paths



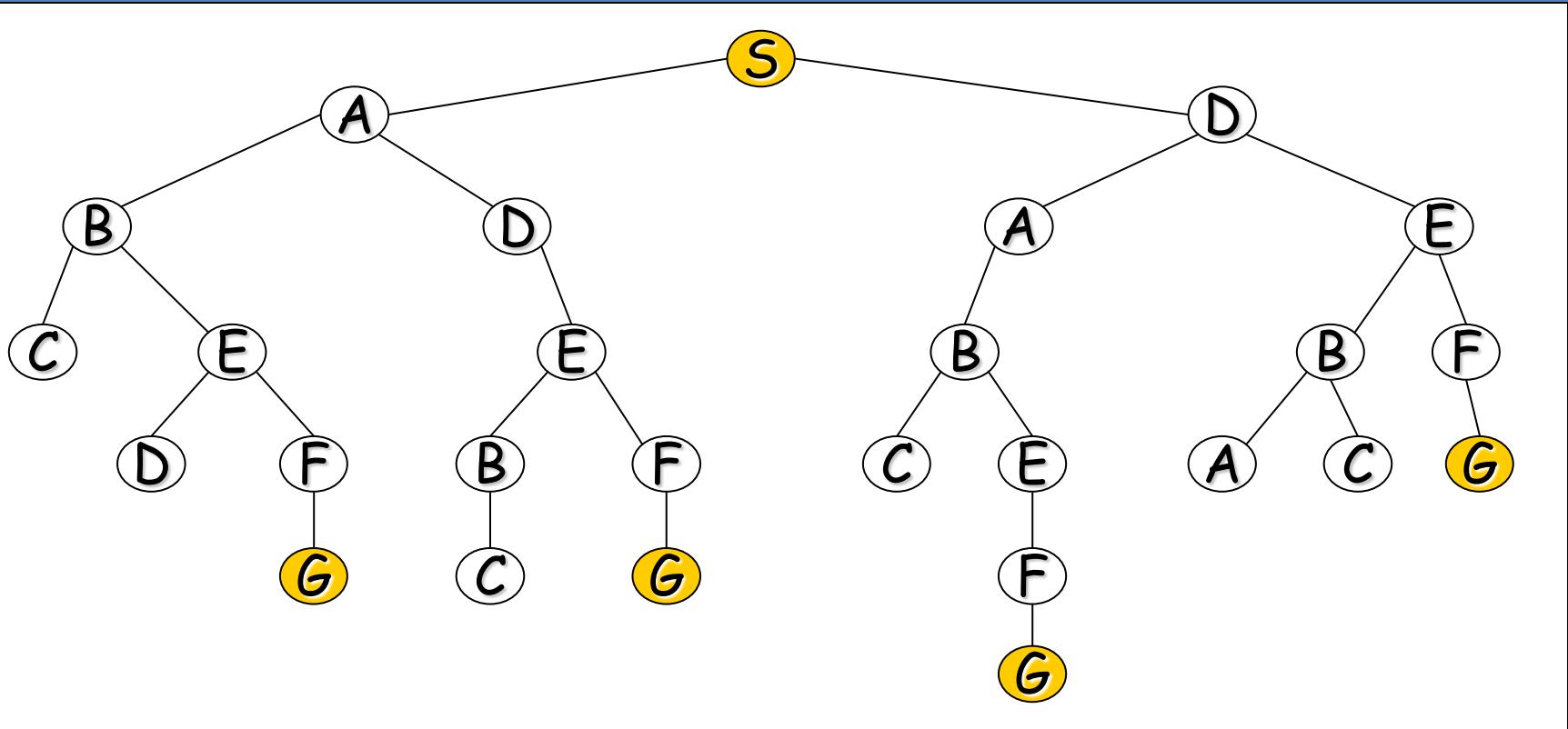
Paths:

- We are not interested in optimal paths here, so we can drop the costs



Note: Nodes do not denote themselves, but denote the partial path from the root to themselves!!

Terminology:



- Node, link (or edge), branch, arc
- Parent, child, ancestor, descendant
- Root node, goal node
- Expand / Open node / Closed node / Branching factor

Using a Tree – The Obvious Solution?

- **But**
 - It can be wasteful on space
 - It can be difficult to implement, particularly if there are varying number of children (as in tic-tac-toe)
 - It is not always obvious which node to expand next
 - We may have to search the tree looking for the best **leaf node** (sometimes called the **fringe** or **frontier** nodes). This can obviously be computationally expensive

How Good is a Solution?

- Does our search method actually find a solution?
- Is it a good solution?
 - Path Cost
 - Search Cost (Time and Memory)
- Does it find the optimal solution?
 - But what is optimal?

Evaluating a Search

- **Completeness**
 - ▣ Is the strategy guaranteed to find a solution?
- **Time Complexity**
 - ▣ How long does it take to find a solution?
- **Space Complexity**
 - ▣ How much memory does it take to perform the search?
- **Optimality**
 - ▣ Does the strategy find the optimal solution where there are several solutions?

Search Trees

- Some issues:
 - Search trees grow very quickly
 - The size of the search tree is governed by the branching factor
 - Even this simple game tic-tac-toe has a complete search tree of 984,410 potential nodes
 - The search tree for chess has a branching factor of about 35

Exercise

1. How are problems solved in artificial intelligence?
2. What is searching?
3.
 - (a) What are the things that could specify a search problem?
 - (b) Supposing you had a robot that is supposed to maneuver it self on a factory floor cluttered with numerous machines and boxes containing both raw and finished materials from the back to the front of the factory. What would be specified for the case in (a)

Search Techniques

- Uninformed Search – Blind Search
- Informed Search – Heuristic Search

Part 3.1 – UNINFORMED (BLIND) SEARCH

We Shall Discuss

- What is Uninformed (Blind) Search?
- Uninformed Search Methods
 - Depth-first search
 - Breadth-first search
 - Non-deterministic search
 - Iterative deepening search
 - Bi-directional search

Uninformed Search?

- Simply searches the state space (or NET)
- Can only distinguish between goal state and non-goal state
- Sometimes called **Blind search** as it has no information or knowledge about its domain

Uninformed Search Characteristics

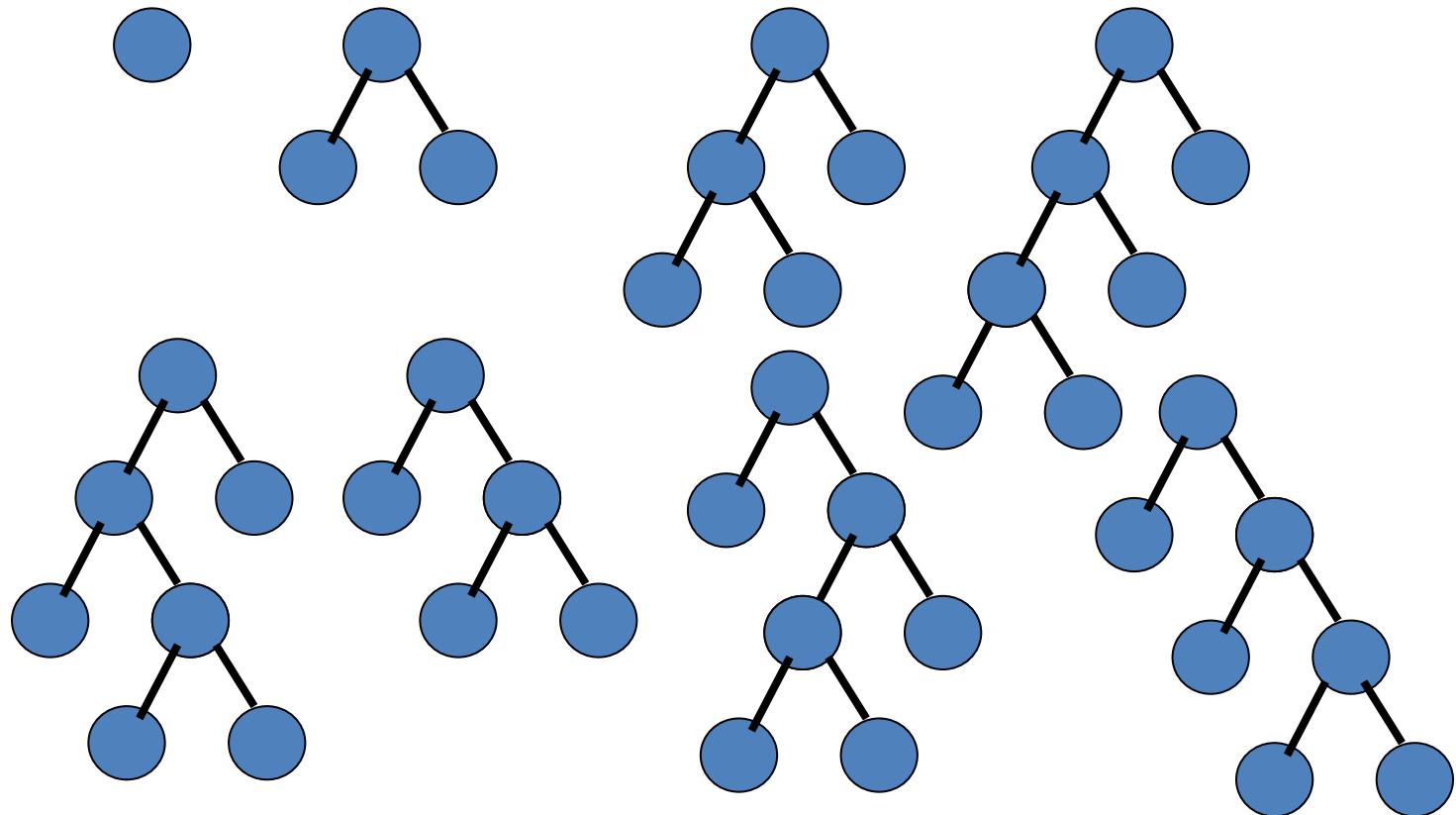
- Blind Searches have **no preference** as to which state (node) that is expanded next
- The different **types** of blind searches are **characterised by the order** in which **they expand** the nodes
 - ▣ This can have a dramatic effect on how well the search performs when measured against the four criteria we defined in an earlier lecture
 - ▣ Search evaluation criteria - Completeness, Time Complexity, Space Complexity, Optimality (of given solution when there are several solutions to choose from)

Uninformed (Blind) Search Methods

- Methods that do not use any specific knowledge about the problem
- These are:
 - ▣ Depth-first search
 - ▣ Breadth-first search
 - ▣ Non deterministic search
 - ▣ Iterative deepening search
 - ▣ Bi-directional search

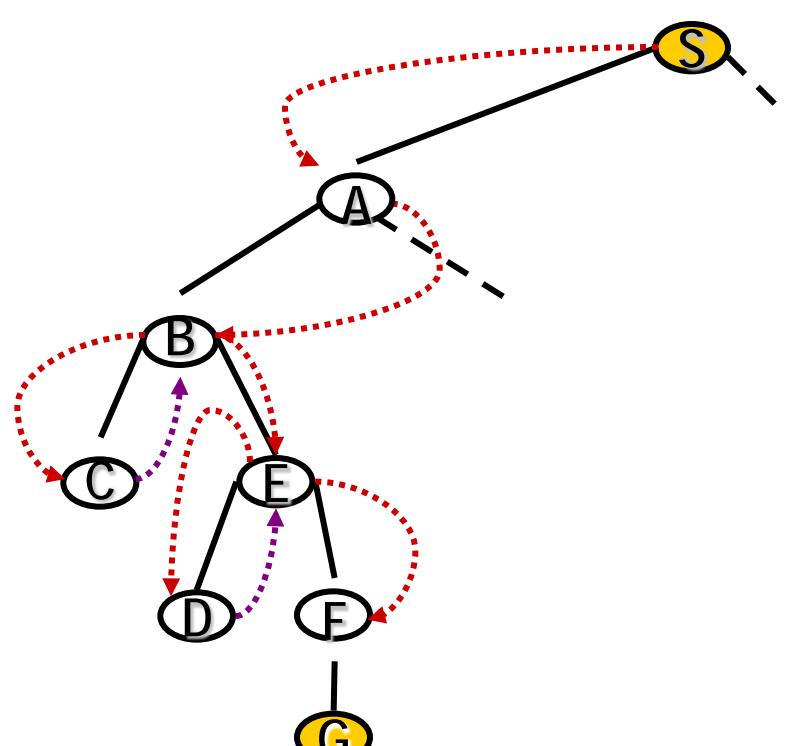
1. Depth-first Search

- Expand the tree as deep as possible, returning to upper levels when needed



Depth-First Search

= Chronological backtracking



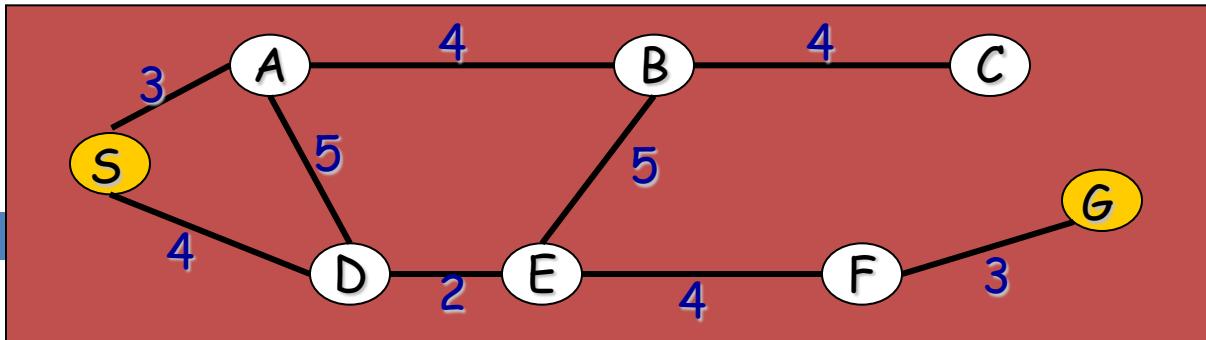
- Select a child
 - convention: left-to-right
- Repeatedly go to next child, as long as possible
- Return to left-over alternatives (higher-up) only when needed

Depth-First Algorithm:

1. **QUEUE** \leftarrow path only containing the root;
2. **WHILE** { **QUEUE** is not empty
 AND goal is not reached

DO { remove the first path from the **QUEUE**;
 create new paths (to all children);
 reject the new paths with loops;
 add the new paths to front of **QUEUE**;

 
3. **IF** goal reached
 THEN success;
 ELSE failure;



1. **QUEUE** \leftarrow path only containing the root;
2. **WHILE** $\begin{cases} \text{QUEUE is not empty} \\ \text{AND goal is not reached} \end{cases}$
 - DO** $\begin{cases} \text{remove the first path from the QUEUE;} \\ \text{create new paths (to all children);} \\ \text{reject the new paths with loops;} \\ \text{add the new paths to front of QUEUE;} \end{cases}$
3. **IF** goal reached
THEN success;
ELSE failure;

Trace of Depth-First for running example:

- (S) S removed, (SA,SD) computed and added
- (SA, SD) SA removed, (SAB,SAD,SAS) computed,
SAB,SAD) added
- (SAB,SAD,SD) SAB removed, (SABA,SABC,SABE) computed,
(SABC,SABE) added
- (SABC,SABE,SAD,SD) SABC removed, (SABCB) computed,
nothing added
- (SABE,SAD,SD) SABE removed, (SABEB,SABED,SABEF)
computed, (SABED,SABEF)added
- (SABED,SABEF,SAD,SD) SABED removed,
(SABEDS,SABEDA,SABEDE) computed,
nothing added
- (SABEF,SAD,SD) SABEF removed, (SABFE,SABFG)
computed, (SABFG) added
- (SABFG,SAD,SD) goal is reached: reports success

Evaluation Criteria:

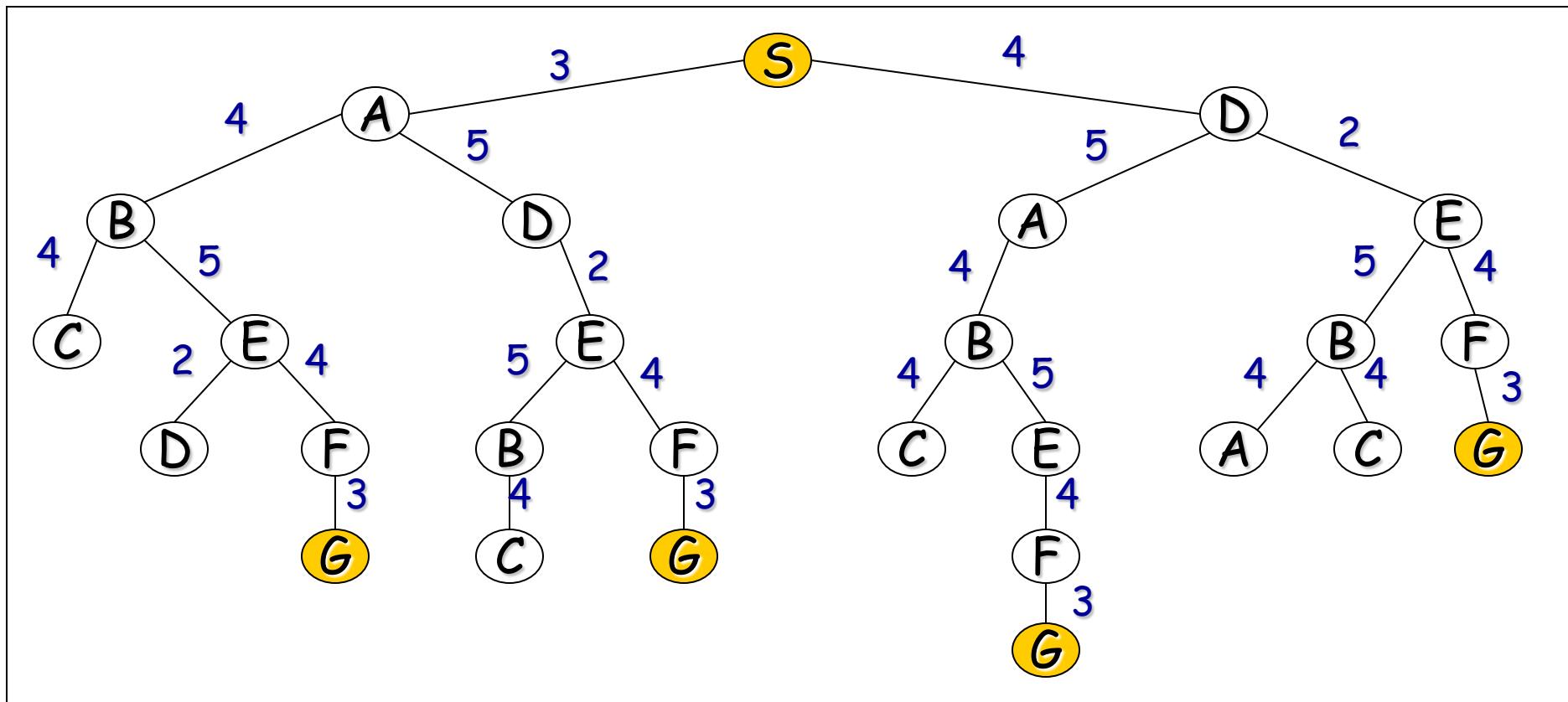
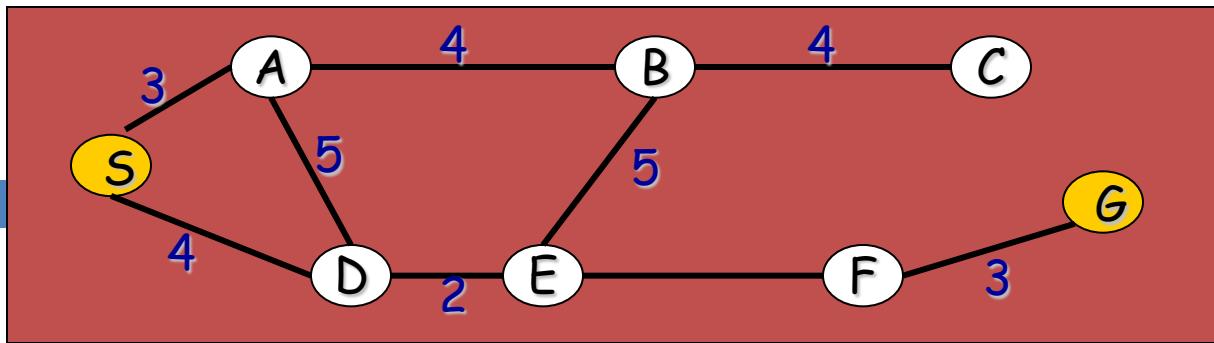
- **Completeness**
 - Does the algorithm always find a path?
 - (for every state space such that a path exists)
- **Speed** (worst time complexity) :
 - What is the highest number of nodes that may need to be created?
- **Memory** (worst space complexity) :
 - What is the largest amount of nodes that may need to be stored?
- Expressed in terms of:
 - d = depth of the tree
 - b = (average) branching factor of the tree
 - m = depth of the shallowest solution

Note: approximations !!

- In our complexity analysis, we do not take the built-in **loop-detection** into account
- The results only ‘formally’ apply to the variants of our algorithms **WITHOUT** loop-checks
- Studying the effect of the loop-checking on the complexity is hard:
 - The overhead of the checking MAY or MAY NOT be compensated by the reduction of the size of the tree
- Also: our analysis **DOES NOT** take the length (space) of representing paths into account !!

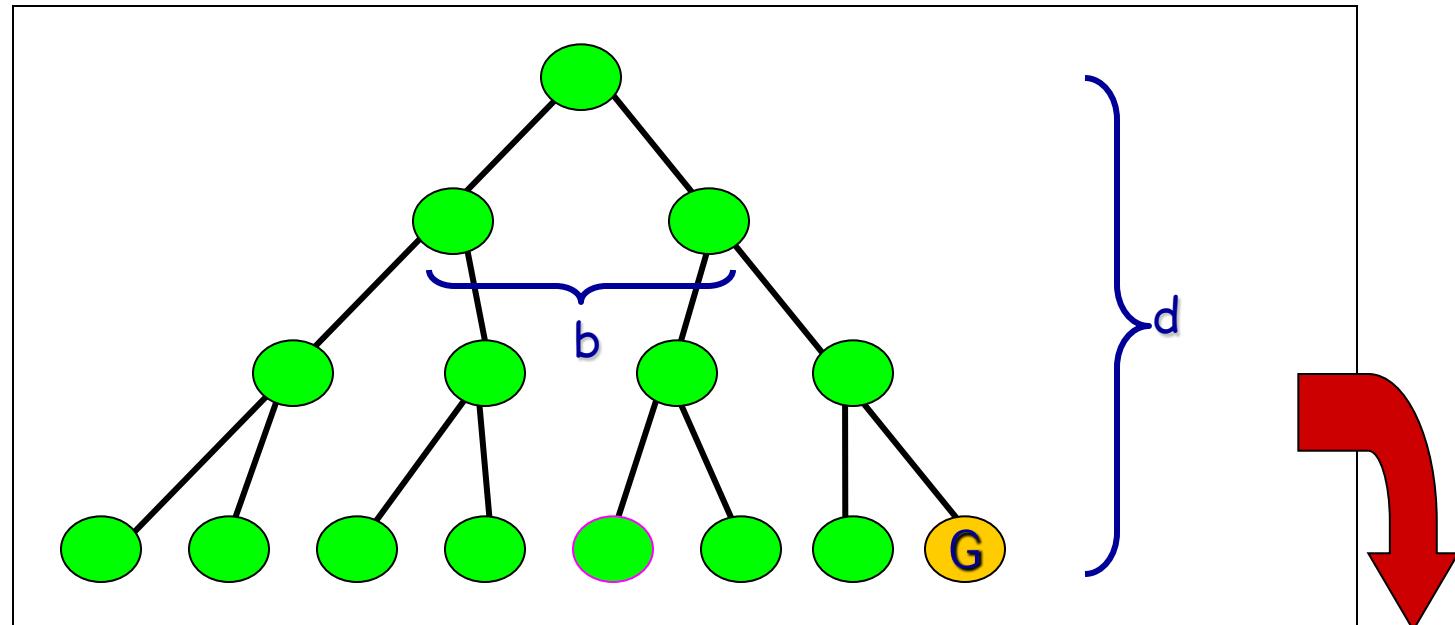
Completeness (depth-first)

- Complete for FINITE (implicit) NETS
 - ▣ (= State space with finitely many nodes)
- **IMPORTANT:**
 - ▣ This is due to integration of LOOP-checking in this version of Depth-First (and in all other algorithms that will follow) !
 - IF we do not remove paths with loops, then Depth-First is not complete (may get trapped in loops of a finite State space)
- **Note:** does NOT find the shortest path



Speed (depth-first)

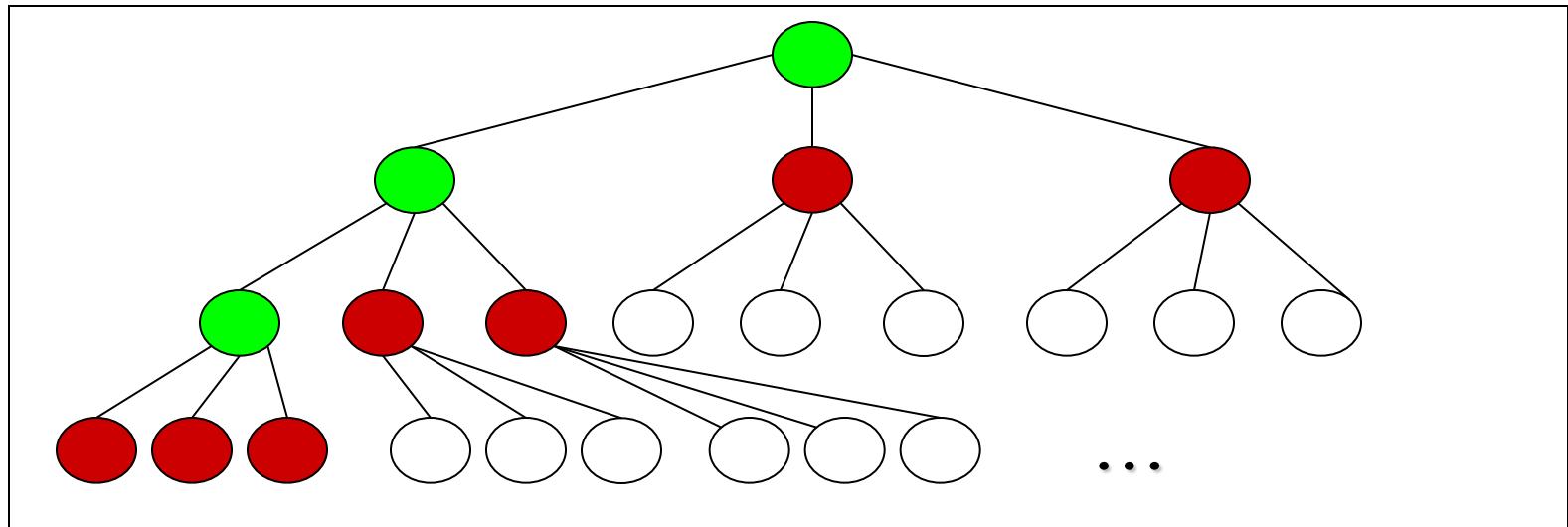
- In the worst case:
 - the (only) goal node may be on the right-most branch,



- Time complexity $= b^d + b^{d-1} + \dots + 1 = \frac{b^{d+1} - 1}{b - 1}$
- Thus: $O(b^d)$

Memory (depth-first)

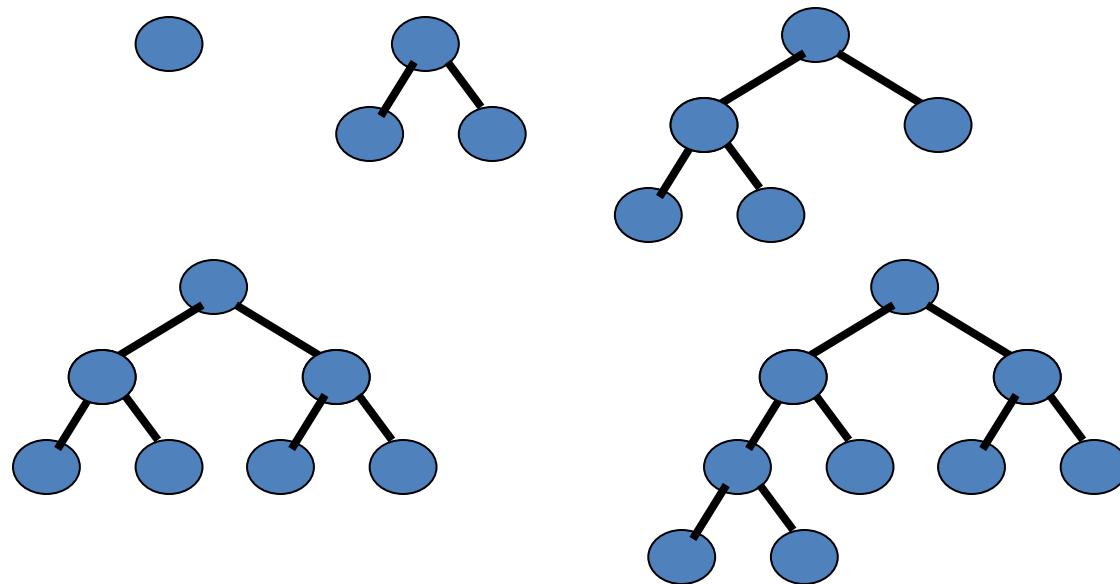
- Largest number of nodes in QUEUE is reached in bottom left-most node
- Example: $d = 3, b = 3$:



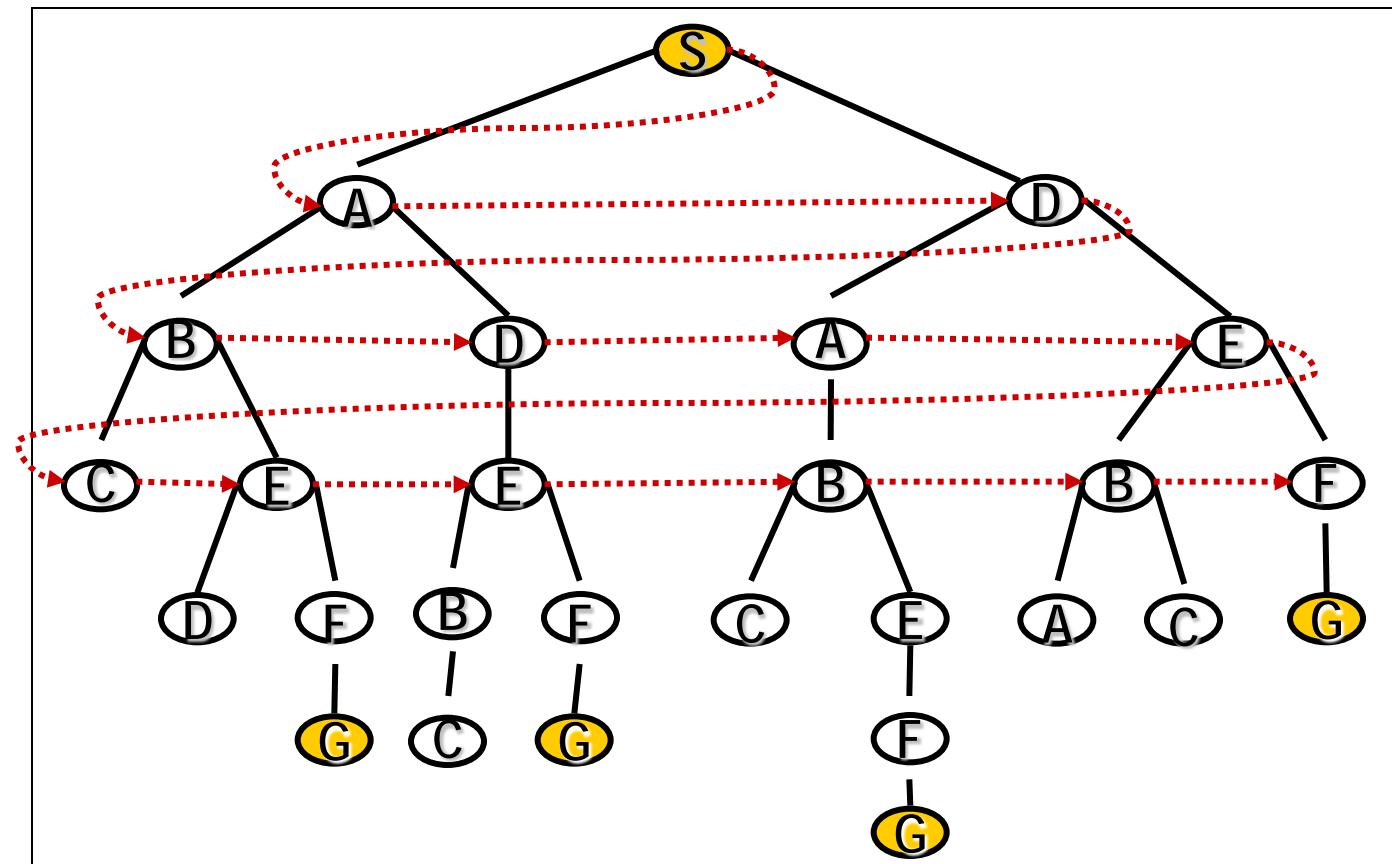
- QUEUE contains all nodes. Thus: 7.
- In General: $((b-1) * d) + 1$
- Order: $O(d*b)$

2. Breadth-First Search

- Expand the tree layer by layer, progressing in depth.
 - In other words,
 - Expand root node first
 - Expand all nodes at level 1 before expanding level 2
- OR
- Expand all nodes at level d before expanding nodes at level $d+1$



Breadth-First Search:



□ Move downwards, level by level, until goal is reached

Breadth-First Algorithm:

1. **QUEUE** \leftarrow path only containing the root;
 2. WHILE { **QUEUE** is not empty
AND goal is not reached

DO { remove the first path from the **QUEUE**;
create new paths (to all children);
reject the new paths with loops;
add the new paths to back of **QUEUE**;
 3. IF goal reached
THEN success;
ELSE failure;
- 
- 
- ONLY
DIFFERENCE !

Trace of breadth-first for running example:

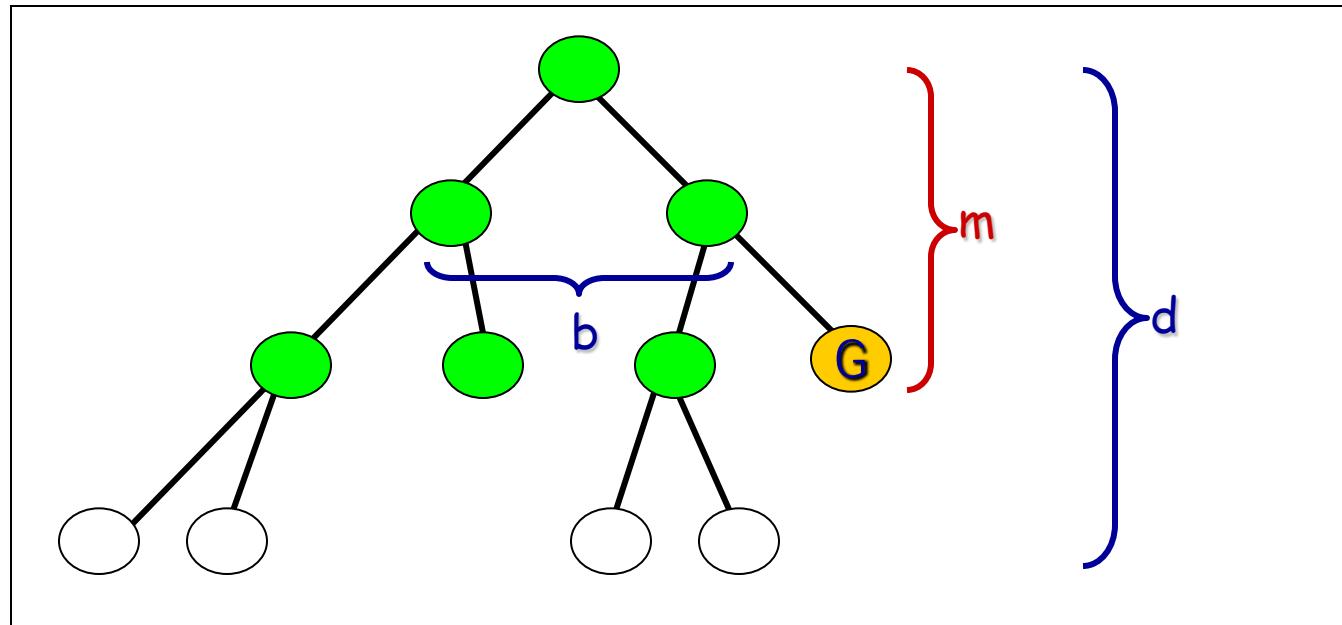
- (S) S removed, (SA,SD) computed and added
- (SA, SD) SA removed, (SAB,SAD,SAS) computed,
(SAB,SAD) added
- (SD,SAB,SAD) SD removed, (SDA,SDE,SDS) computed,
(SDA,SDE) added
- (SAB,SAD,SDA,SDE) SAB removed, (SABA,SABE,SABC)
computed, (SABE,SABC) added
- (SAD,SDA,SDE,SABC,SABE) SAD removed, (SADS,SADA, SADE)
computed, (SADE) added
- etc, until QUEUE contains:
 - (SABED,SABEF,SADEB,SADEF,SDABC,SDABE,SDEBA,SDEBC, SDEFG)
goal is reached: reports success

Completeness (breadth-first)

- **Complete**
 - even for infinite implicit NETS !
 - Would even remain complete without our loop-checking
- **Note:** ALWAYS finds the shortest path

Speed (breadth-first)

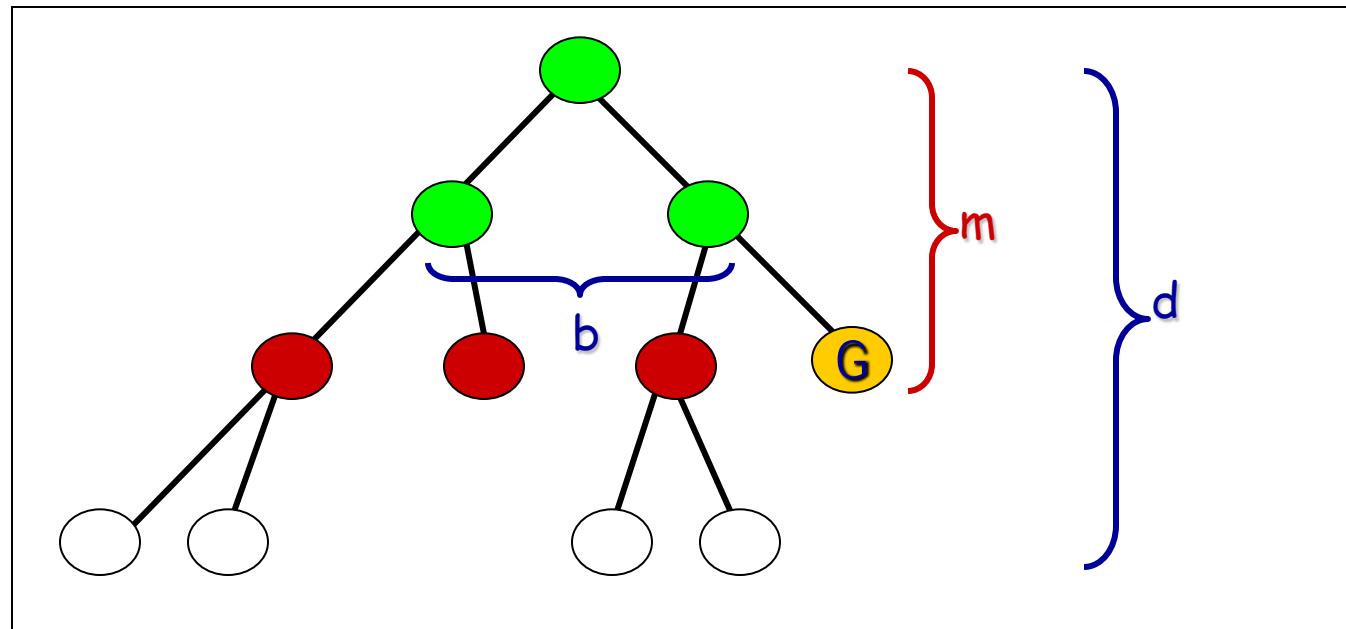
- If a goal node is found on depth m of the tree, all nodes up till that depth are created



- Thus: $O(b^m)$
- **Note:** depth-first would also visit deeper nodes

Memory (breadth-first)

- Largest number of nodes in QUEUE is reached on the level m of the goal node



- QUEUE contains all  and  nodes. (Thus: 4)
- In General: b^m
- This usually is MUCH worse than depth-first !!

Exponential Growth (breadth-first)

Depth	Nodes	Time		Memory	
0	1	1	millisecond	100	kbytes
2	111	0.1	second	11	kilobytes
4	11,111	11	seconds	1	megabyte
6	10^6	18	minutes	111	megabytes
8	10^8	31	hours	11	gigabytes
10	10^{10}	128	days	1	terabyte
12	10^{12}	35	years	111	terabytes
14	10^{14}	3500	years	11,111 terabytes	

- Time and memory requirements for breadth-first search, assuming a branching factor of 10, 100 bytes per node and searching 1000 nodes/second

Exponential Growth - Breadth-First Observations

- Space is more of a factor to breadth first search than time
- Time is still an issue. Who has 35 years to wait for an answer to a level 12 problem (or even 128 days to a level 10 problem)
- It could be argued that as technology gets faster then exponential growth will not be a problem. But even if technology is 100 times faster we would still have to wait 35 years for a level 14 problem and what if we hit a level 15 problem!

Practical Evaluation:

- **1. Depth-first search:**
 - IF the search space contains very deep branches without solution, THEN Depth-first may waste much time in them
- **2. Breadth-first search:**
 - Is VERY demanding on memory !
- **Solutions ??**
 - Non-deterministic search
 - Iterative deepening

3. Non-deterministic Search

- A Non-deterministic algorithm is an algorithm that, even for the same input, can exhibit different behaviors on different runs, as opposed to a deterministic algorithm
- There are several ways an algorithm may behave differently from run to run

Non-deterministic Search

1. **QUEUE** \leftarrow path only containing the root;
 2. WHILE { **QUEUE** is not empty
 AND goal is not reached

DO { remove the first path from the **QUEUE**;
 create new paths (to all children);
 reject the new paths with loops;
 add the new paths in random places in **QUEUE**;
 3. IF goal reached
 THEN success;
 ELSE failure;
- 
- 

4. Iterative Deepening Search

- Also referred to as **Iterative Deepening Depth-First Search**
- Restrict a depth-first search to a fixed depth
 - a depth-limited version of depth-first search is run repeatedly with increasing depth limits until the goal is found
- If no path is found, increase the depth and restart the search

Depth-limited Search

1. DEPTH <-- <some natural number>
QUEUE <-- path only containing the root;
2. WHILE { QUEUE is not empty
AND goal is not reached
DO { remove the first path from the QUEUE;
~~IF~~ path has length smaller than DEPTH
 create new paths (to all children);
 reject the new paths with loops;
 add the new paths to front of QUEUE;
3. IF goal reached
 THEN success;
 ELSE failure;

Iterative Deepening Algorithm:

1. DEPTH \leftarrow 1

2. WHILE goal is not reached

DO { perform Depth-limited search;
 { increase DEPTH by 1;

Iterative Deepening: the best 'blind' search

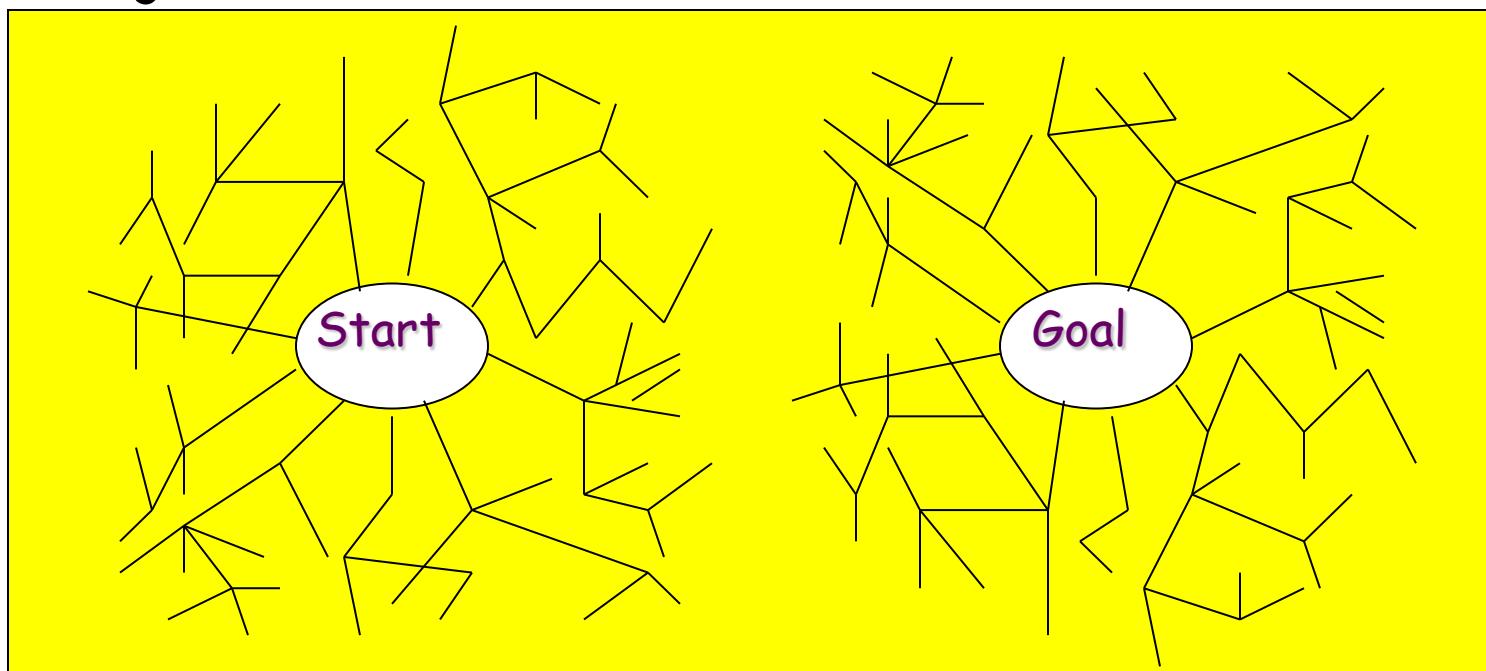
- Complete: yes - even finds the shortest path (like breadth first)
- Memory: b^m (combines advantages of depth- and breadth-first)
- Speed:
 - If the path is found for Depth = m , then how much time was wasted constructing the smaller trees??
- $b^{m-1} + b^{m-2} + \dots + 1 = \frac{b^m - 1}{b - 1} = O(b^{m-1})$
- While the work spent at DEPTH = m itself is $O(b^m)$



In general: **VERY good trade-off**

5. Bi-directional Search

- Compute the tree from the start node and from a goal node, until these meet
- **IF** you are able to EXPLICITLY describe the GOAL state, **AND** you have BOTH rules for FORWARD reasoning AND BACKWARD reasoning:



Bi-directional Algorithm:

1. $\text{QUEUE1} \leftarrow$ path only containing the root;
 $\text{QUEUE2} \leftarrow$ path only containing the goal;
2. WHILE both QUEUE_i are not empty
AND QUEUE1 and QUEUE2 do NOT share a state

DO {
remove their first paths;
create their new paths (to all children);
reject their new paths with loops;
add their new paths to back;
}
3. IF QUEUE1 and QUEUE2 share a state
THEN success;
ELSE failure;

Properties (Bi-directional):

- Complete: Yes.
- Speed: If the test on common state can be done in constant time (hashing):
 - $2 * O(b^{m/2}) = O(b^{m/2})$
- Memory: similarly: $O(b^{m/2})$

Part 3.2 – INFORMED (HEURISTIC) SEARCH

We Shall Discuss

- Concept
- Informed search methods

Search with Domain Knowledge added

- Uninformed (blind) searches are normally very inefficient
- Adding domain knowledge can improve the search process
- Concept of Informed (Heuristic) Search
 - Heuristic (informed) search → explore the node that is most “likely” to be the nearest to a goal state
 - There is no guarantee that the heuristic provided most “likely” node will get you closer to a goal state than any other

Knowledge/info



Examples:

- An office in a city building
 - Find me in Office door number N311 e.g. KICC, NSSF, LU Main Campus, Comp Lab building?
 - Wing, floor, left/right
- Some streets in Nairobi are named in alphabetical order
- Visit the doctor
 - Symptoms: fever, nausea, headache, ...
 - **Leading questions:** how long?, traveled?, ... (Malaria, typhoid, meningitis, flu,..)
 - x Blood test, y test,...

Heuristic Searches

- Characteristics
 - Has some **domain knowledge**
 - Usually **more efficient** than blind searches
 - Also **called informed search**
 - Heuristic searches **work by deciding which is the next best node to expand** (there is no guarantee that it is the best node)
- Why Use?
 - It may be too resource intensive (both time and space) to use a blind search
 - Even if a blind search will work we may want a more efficient search method

Heuristic Search Methods

- Methods that use a heuristic function to provide specific knowledge about the problem:
 - Heuristic Functions
 - Hill climbing
 - Greedy search
 - A* search algorithm

Heuristic Functions

- To further improve the quality of the previous methods, we need to include **problem-specific knowledge on the problem**
 - How can this be done in such a way that the algorithms remain generally applicable ???

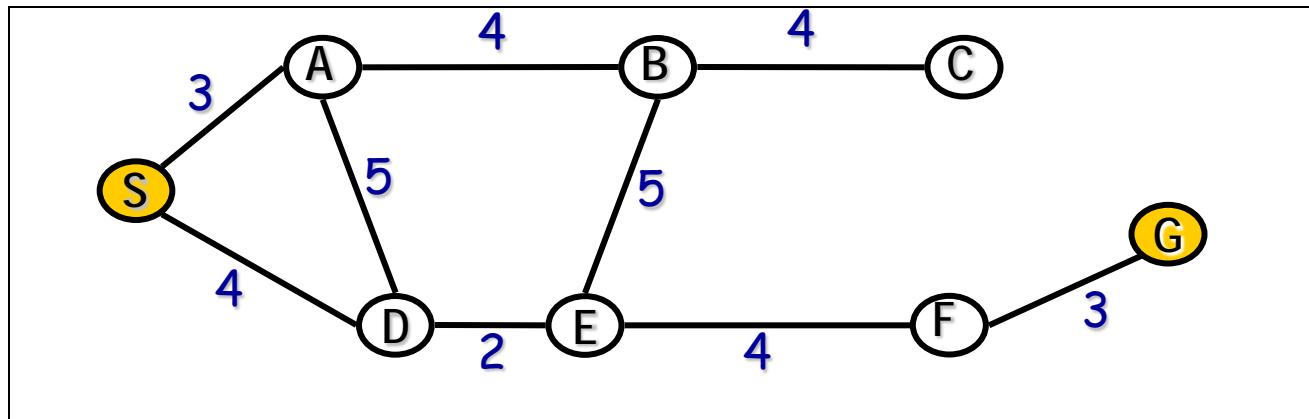
- HEURISTIC FUNCTIONS:
 - h : States → Numbers
 - $h(n)$: expresses the quality of the state n
 - allow to express problem-specific knowledge in the search method algorithm

Heuristic Functions

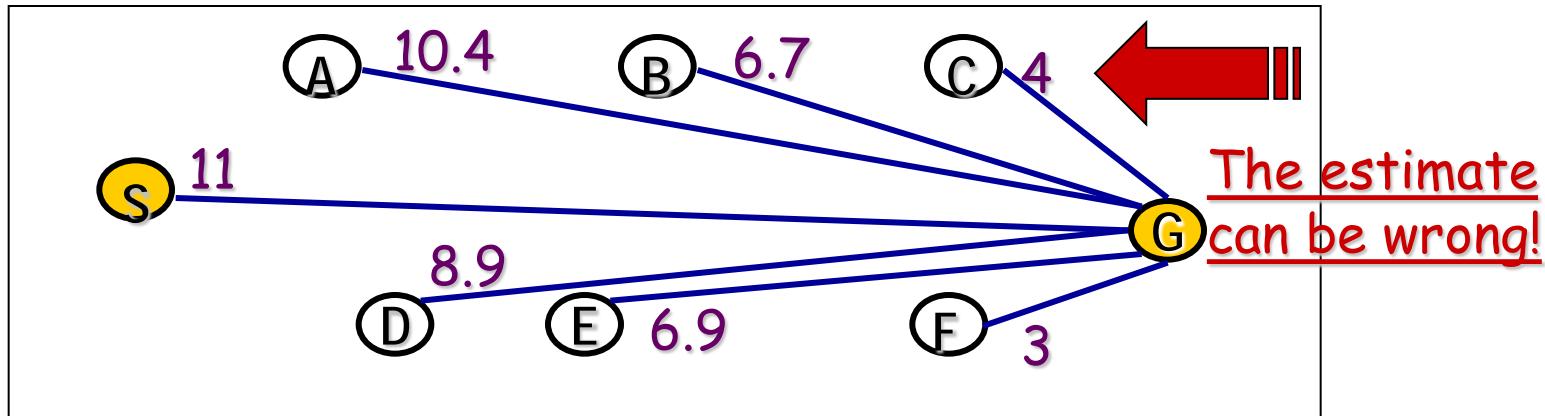
- **Heuristic function $h(n)$** , takes a node n and returns a non-negative real number that is an estimate of the path cost from node n to a goal node
- The heuristic function is a way to inform the search about the direction to a goal
- It provides an informed way to guess which neighbor of a node will lead to a goal

Example 1: road map

- Imagine the problem of finding a route on a road map and that the NET below is the road map:



- Define $h(n) =$ the straight-line distance from n to G



Example 2: 8-puzzle

- $h_1(n)$ = the number correctly placed tiles on the board:

$$h_1 \left(\begin{array}{|c|c|c|} \hline 1 & 3 & 2 \\ \hline 8 & & 4 \\ \hline 5 & 6 & 7 \\ \hline \end{array} \right) = 4$$

- $h_2(n)$ = number of incorrectly placed tiles on board:
 - gives (rough!) estimate of how far we are from goal

$$h_2 \left(\begin{array}{|c|c|c|} \hline 1 & 3 & 2 \\ \hline 8 & & 4 \\ \hline 5 & 6 & 7 \\ \hline \end{array} \right) = 4$$

Most often, 'distance to goal' heuristics are more useful !

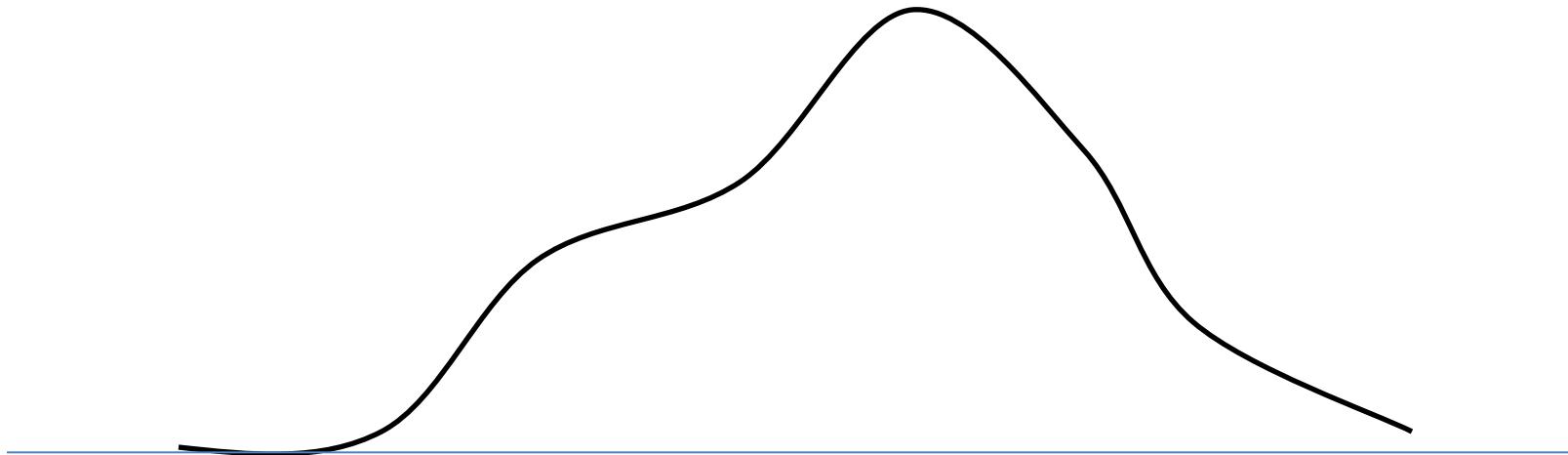
Example 2: 8-puzzle Manhattan distance

- $h_3(n)$ = the sum of (the horizontal + vertical distance that each tile is away from its final destination):
 - gives a better estimate of distance from the goal node

$$h_3 \left(\begin{array}{|c|c|c|} \hline 1 & 3 & 2 \\ \hline 8 & & 4 \\ \hline 5 & 6 & 7 \\ \hline \end{array} \right) = 1 + 1 + 2 + 2 = 6$$

Heuristic searches- Hill climbing

- A basic heuristic search method:
 - depth-first + heuristic

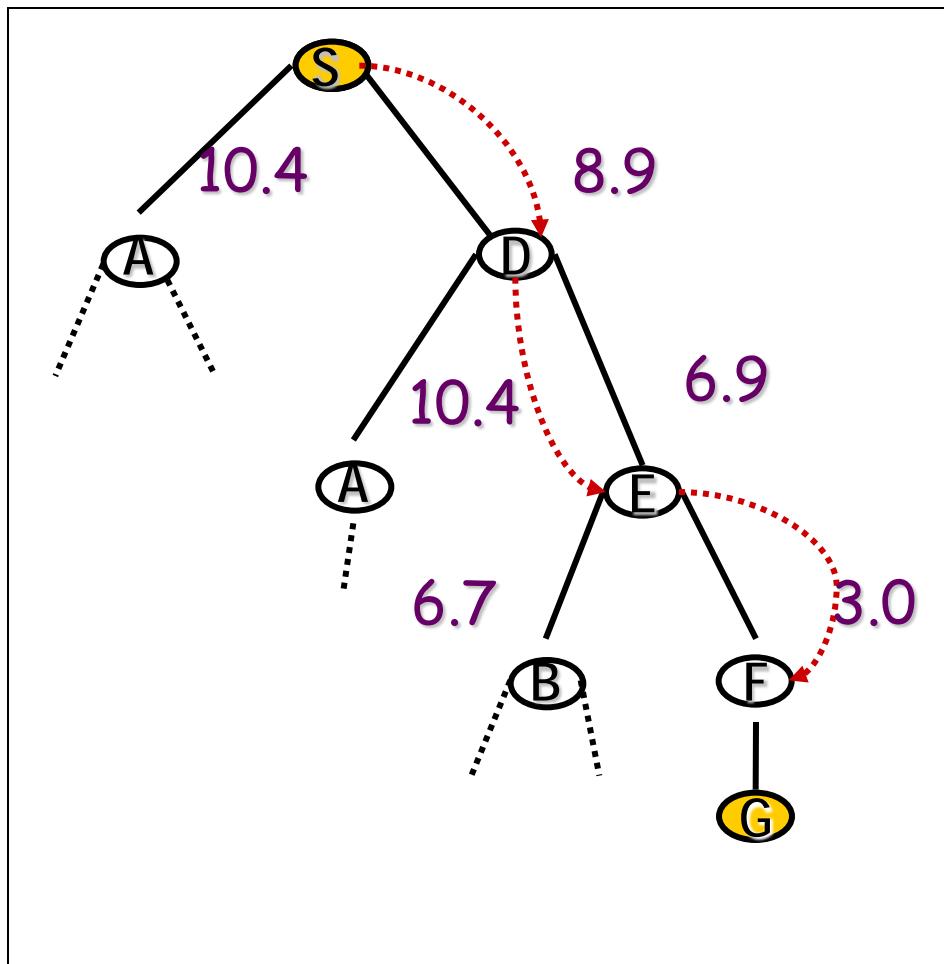


Hill climbing

- Described as: "Like climbing Everest in thick fog with amnesia"
- It is a mathematical optimization technique which belongs to the family local search
 - The most basic of all optimization techniques is evolution
- **Hill climbing** is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution
- If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found

Hill climbing_1

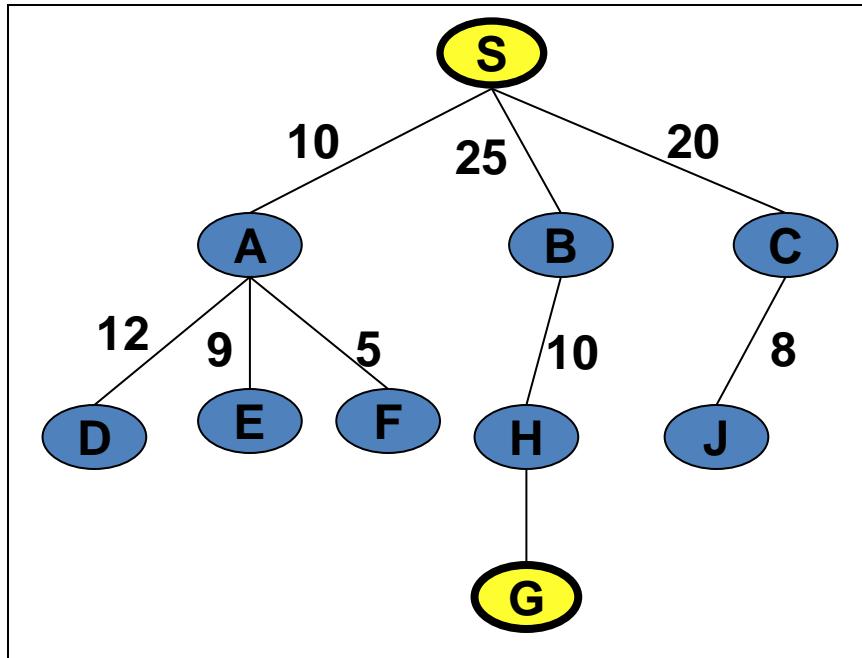
- Example: using the straight-line distance:



- Perform depth-first,
BUT:
 - instead of left-to-right selection,
 - FIRST select the child with the best heuristic value
- Note: We are applying **minimal cost search** in this example

Minimal Cost Search Example

- Example: using the heuristic value



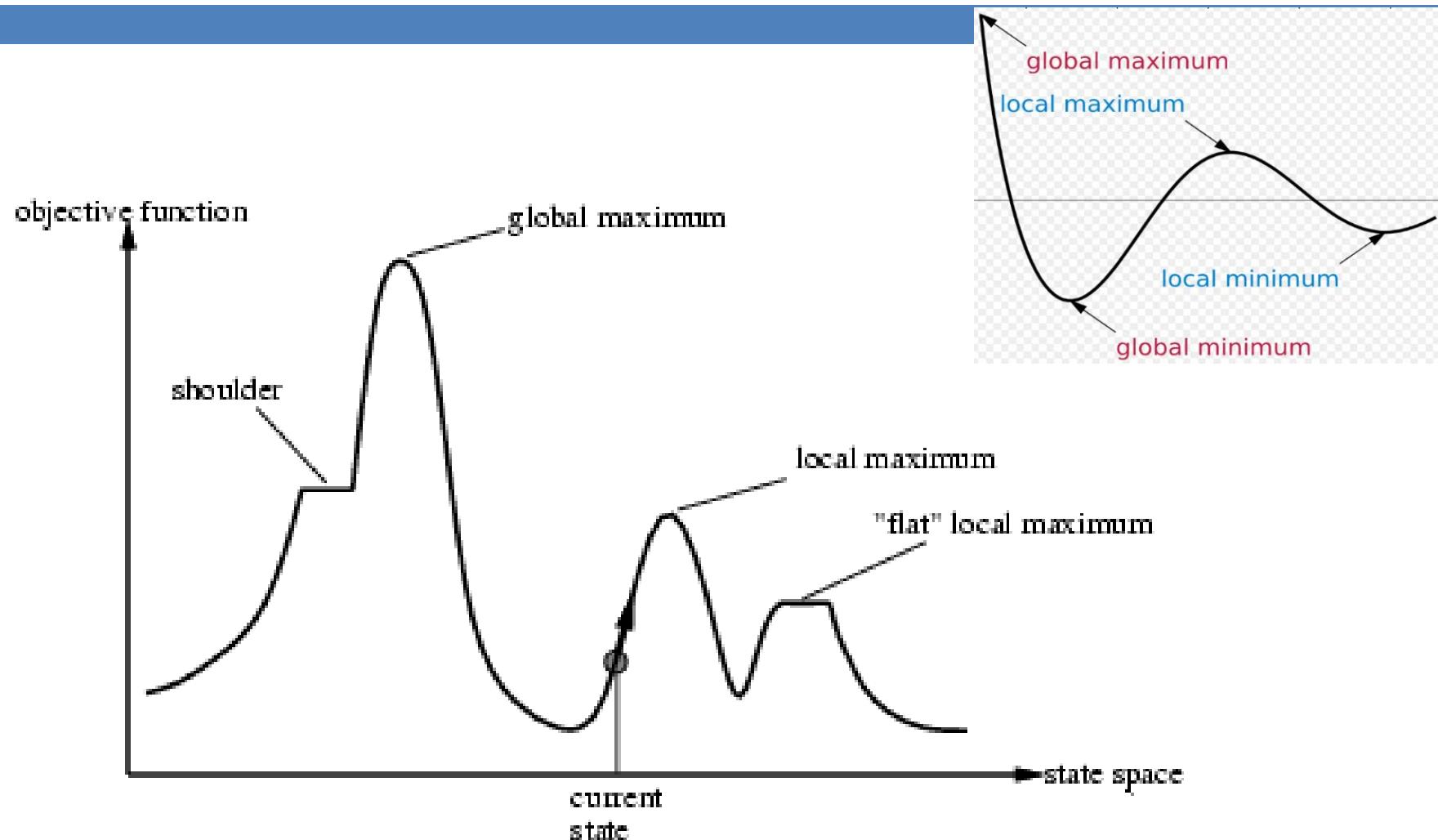
- If p is the current path:
 - the next path extends p by adding the node with the smallest cost from the endpoint of p

S
SA(10)
SAF(5)
...
.....

Hill climbing_2

- Inspiring Example: climbing a hill in the fog.
 - Heuristic function: check the change in altitude in 4 directions: the strongest increase is the direction in which to move next
- Is identical to Hill climbing_1, except for dropping the backtracking(loops)
- Produces a number of classical problems:
 - depending on initial state, can get stuck in local maxima

Problems with Hill climbing_2:



Problems with Hill climbing_2:

- Hill climbing cannot reach the optimal/best state(global maximum) if it enters any of the following regions :
 - ▣ **Local maximum** : At a local maximum all neighboring states have a values which is worse than the current state. Since hill climbing uses greedy approach, it will not move to the worse state and terminate itself. The process will end even though a better solution may exist.
 - ▣ To overcome local maximum problem : Utilize backtracking technique. Maintain a list of visited states. If the search reaches an undesirable state, it can backtrack to the previous configuration and explore a new path.
 - ▣ **Plateau** : On plateau all neighbors have same value . Hence, it is not possible to select the best direction.
 - ▣ To overcome plateaus : Make a big jump. Randomly select a state far away from current state. Chances are that we will land at a non-plateau region
 - ▣ **Ridge** : Any point on a ridge can look like peak because movement in all possible directions is downward. Hence the algorithm stops when it reaches this state.
 - ▣ To overcome Ridge : In this kind of obstacle, use two or more rules before testing. It implies moving in several directions at once

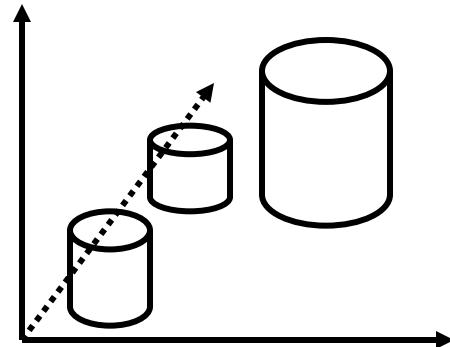
Problems with Hill climbing_2:

Foothills:

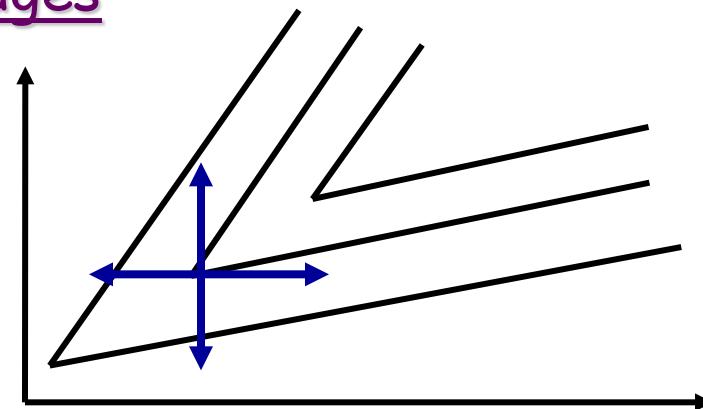
Local maximum



Plateaus:



Ridges:

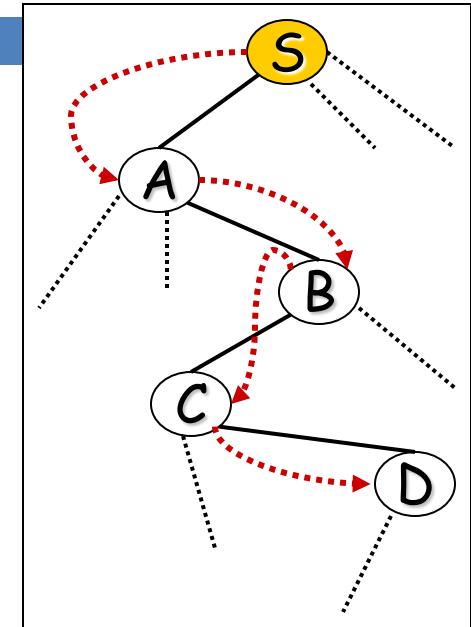


Comments:

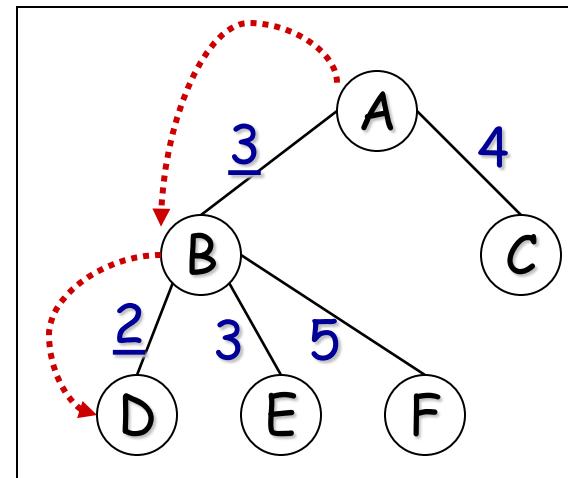
- Foothills are local maxima: it's a state that is better than all its neighbors but is not better than some other states farther away. hill climbing_2 can't detect the difference
- Plateaus is a flat area of the search space in which a whole set of neighboring states have the same value. It is not possible to determine the best direction in which to move and therefore doesn't allow you to progress in any direction
 - Foothills and plateaus require **random jumps** to be combined with the hill climbing algorithm
- Ridges neither: a ridge is a special kind of local maximum. It is an area of the search space that is higher than the surrounding areas and that itself has a slope. Any point on a ridge can look like a peak because the directions you have fixed in advance all move downwards for this surface
 - Ridges require **new rules**, more directly targeted to the goal, to be introduced (new directions to move)

Local search

- Hill climbing_2 is an example of local search.
- In local search, we only keep track of 1 path and use it to compute a new path in the next step.
 - ▣ QUEUE is always of the form:
 - (p)



- Another example:
 - MINIMAL COST search:
- If p is the current path:
 - the next path extends p by adding the node with the smallest cost from the endpoint of p



Heuristic Searches – Best-First Search Algorithms

- Greedy Search algorithm
- A* Search algorithm

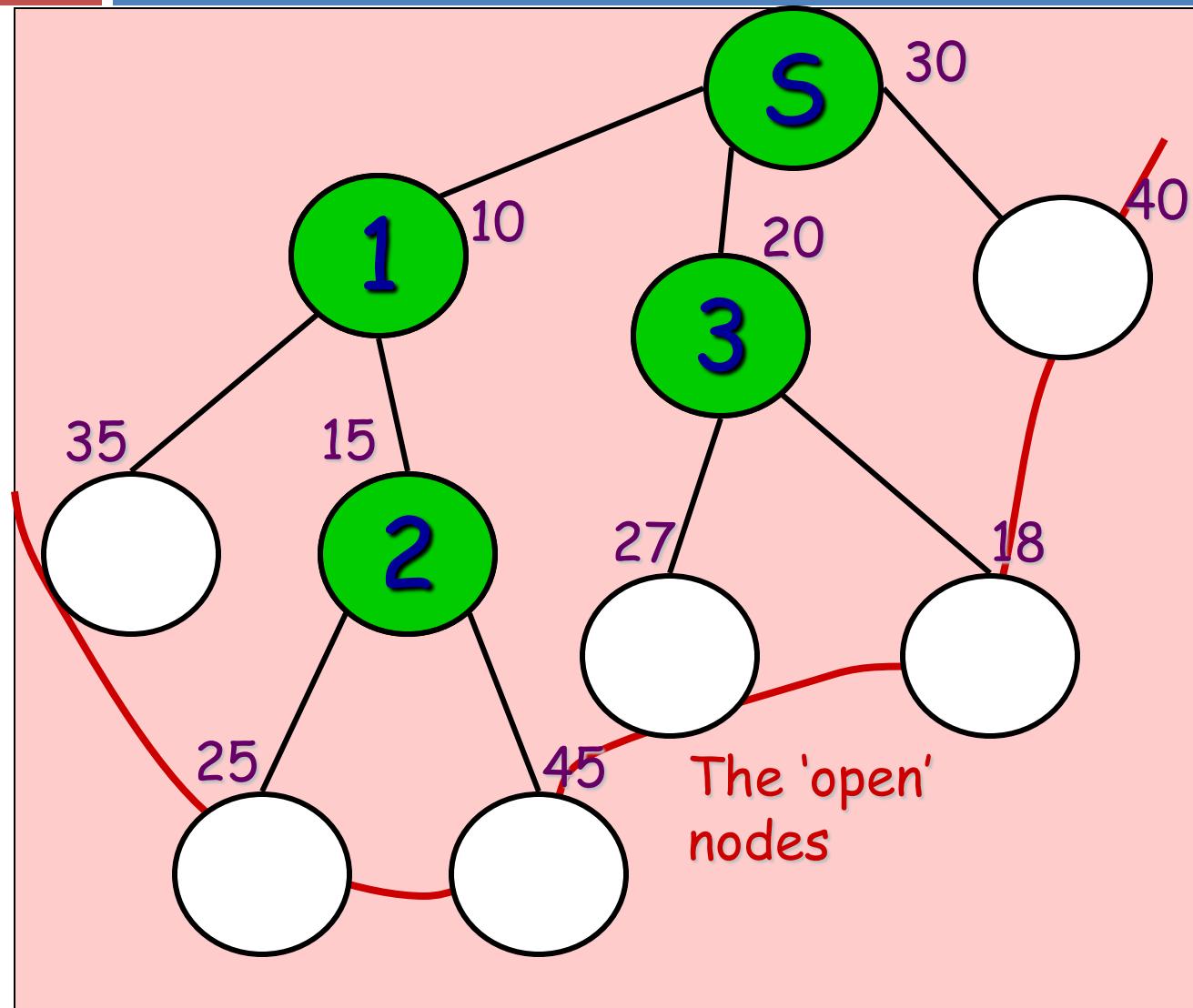
Greedy Search

- So named as it takes the biggest “bite” it can out of the problem
 - ▣ That is, **it seeks to minimise the estimated cost to the goal** by expanding the node estimated to be closest to the goal state

in other words,

- Always expand the heuristically best nodes first

Greedy Search, or Heuristic best-first search:



- At each step, select the node with the best (in this case: lowest) heuristic value

Greedy Search algorithm:

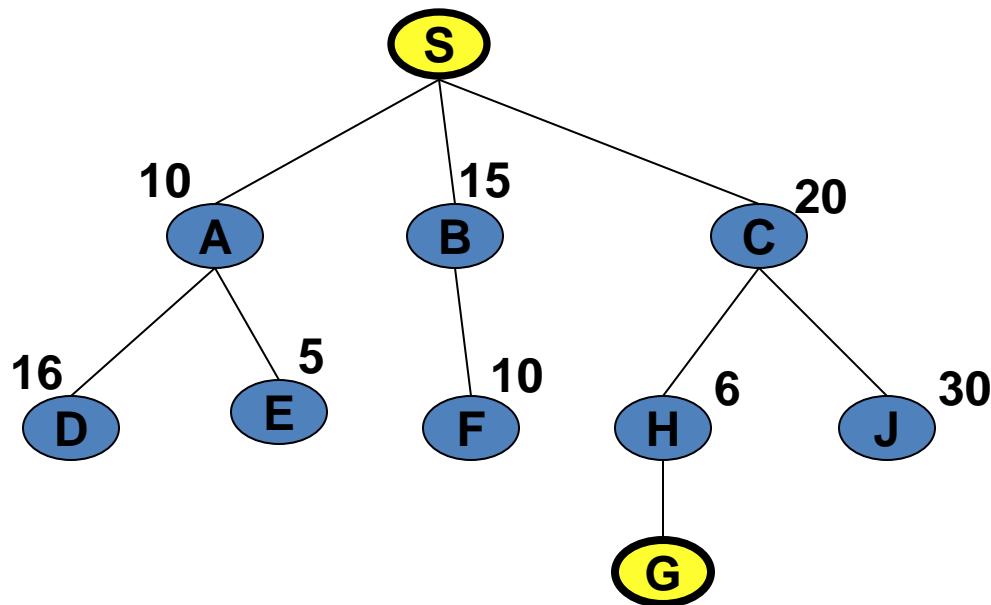
1. **QUEUE** \leftarrow path only containing the root;
2. **WHILE** { **QUEUE** is not empty
 AND goal is not reached

DO { remove the first path from the **QUEUE**;
 create new paths (to all children);
 reject the new paths with loops;
 add the new paths and sort the entire **QUEUE**;
 (HEURISTIC)

3. **IF** goal reached
 THEN success;
 ELSE failure;

Greedy Search example

- Example: using the heuristic value



- At each step, select the node with the best (in this case: lowest) heuristic value

S

SA(10),SB(15),SC(20)

SAD(16),SAE(5),SB(15),SC(20)

SAE(5),SB(15),SAD(16),SC(20)

SAE(5),SB(15),SAD(16),SC(20)

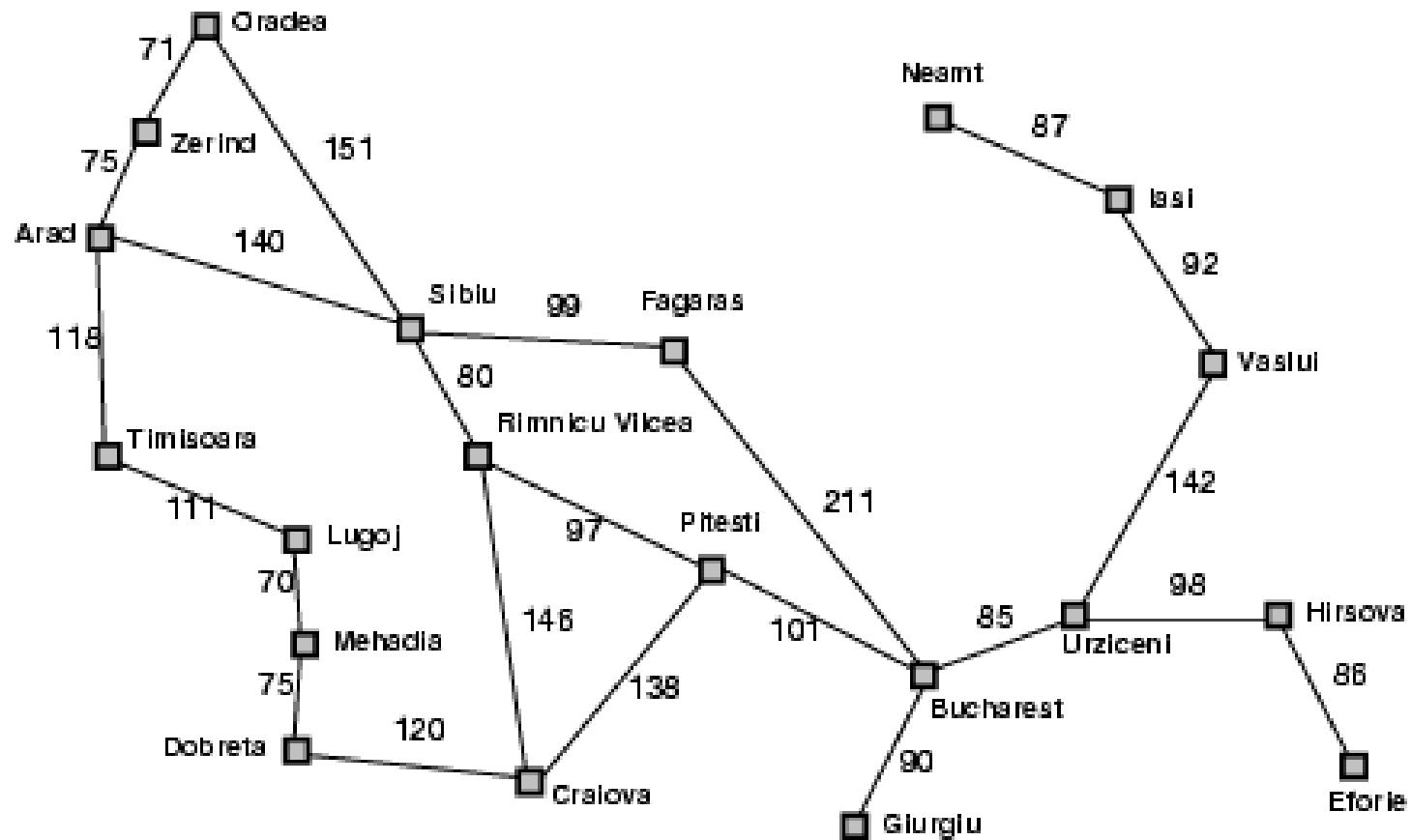
SBF(10),SAD(16),SC(20)

SBF(10),SAD(16),SC(20)

SCH(6),SCJ(30)

SCHG <=goal

Example: Greedy algorithm for Romania Tour with step costs in km



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

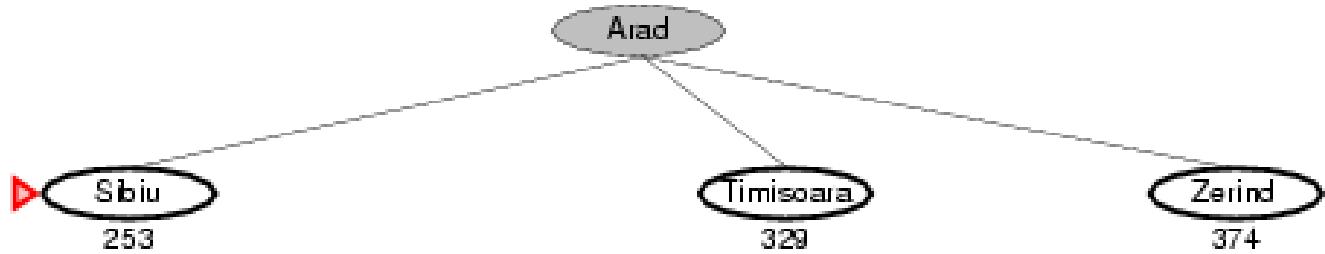
Example cont...

- Evaluation function $f(n) = h(n)$ (**heuristic**)
= estimate of cost from n to goal
- e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest
- Greedy best-first search expands the node that **appears** to be closest to goal

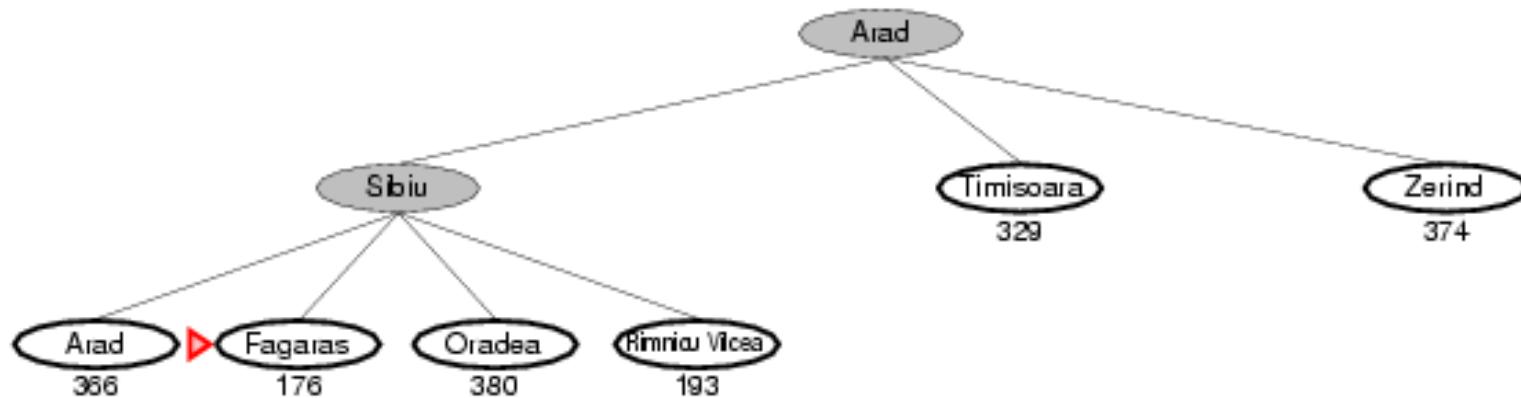
Example cont...



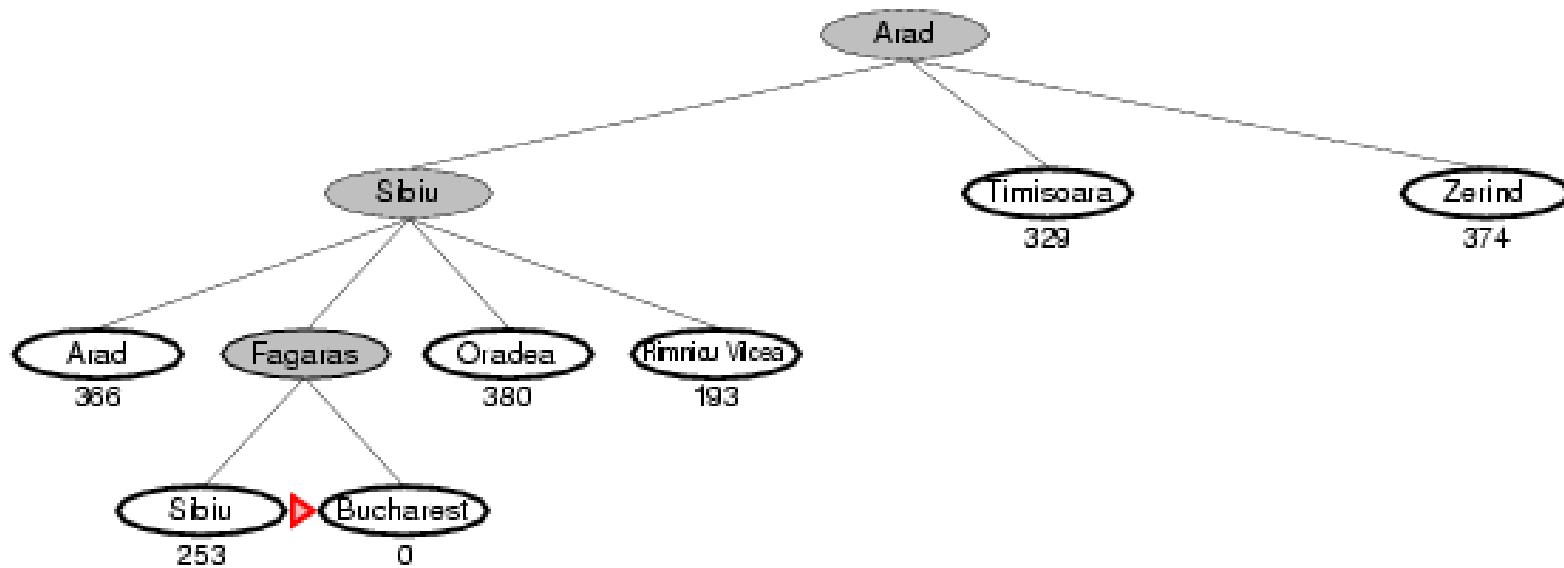
Example cont...



Example cont...



Example cont...



Heuristic Searches - Greedy Search

- It is only concerned with short term aims
- It is not optimal
- It is not complete
 - ▣ It is possible to get stuck in an infinite loop, unless you check for repeated states
 - ▣ e.g., Iasi → Neamt → Iasi → Neamt →
- Time complexity $O(b^m)$, but a good heuristic can give dramatic improvement
- Space complexity $O(b^m)$, keeps all nodes in memory

Heuristic Searches - A* Algorithm

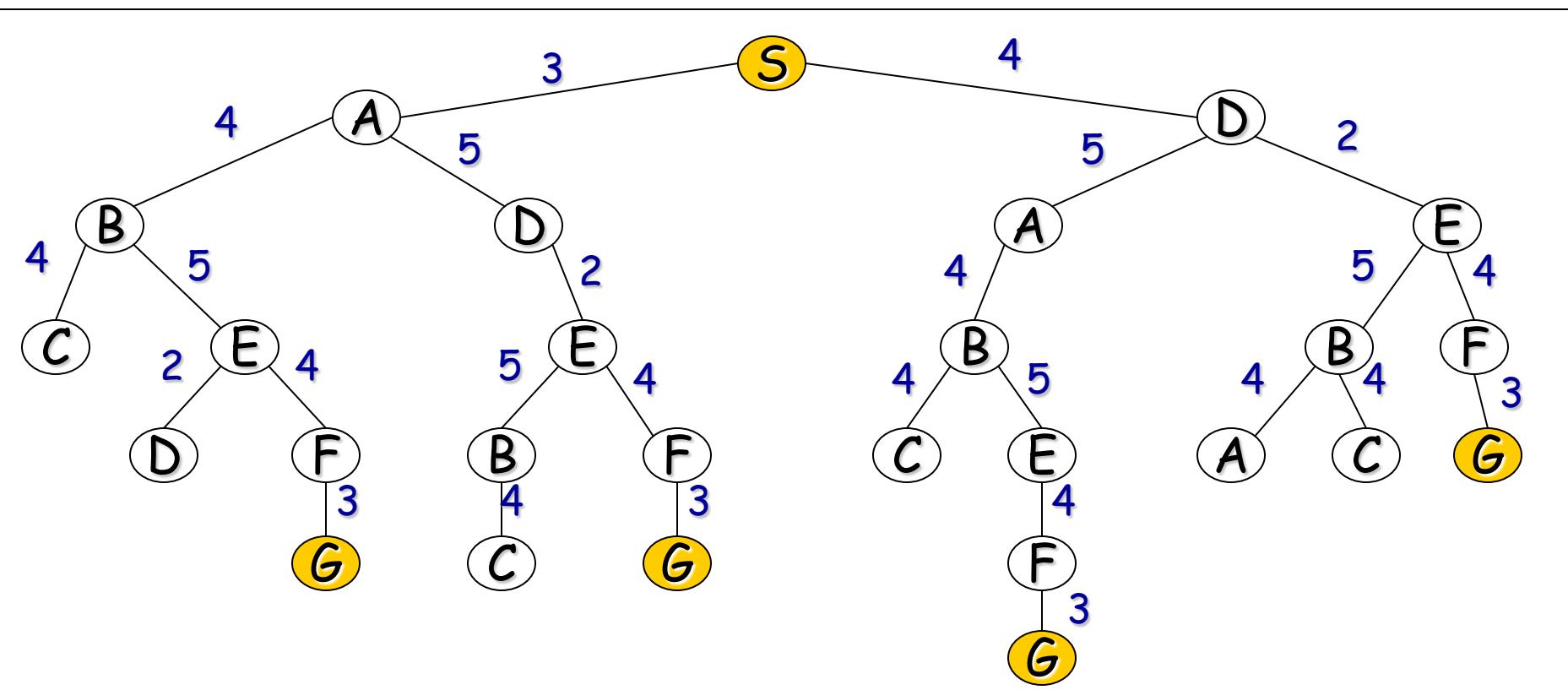
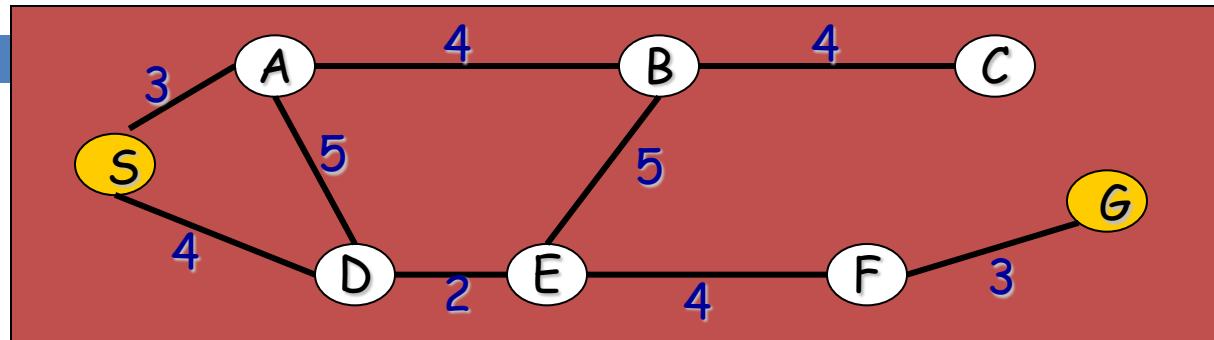
- A combination of Greedy search and Uniform cost search.
 - to compliment one another
- This search method minimises the cost to the goal using an heuristic function, $h(n)$
 - Greedy search can considerably cut the search time but it is neither optimal nor complete
- By comparison uniform cost search minimises the cost of the path so far, $g(n)$
 - Uniform cost search is both optimal and complete but can be very inefficient

Heuristic Searches - A* Algorithm

- Idea: **avoid expanding paths that are already expensive**
- Always expand the path that has a minimum value of $f(n)$
- Evaluation function $f(n) = g(n) + h(n)$
 - $g(n)$ = cost so far to reach n
 - $h(n)$ = estimated cost from n to goal
 - $f(n)$ = estimated total cost of path through n to goal

Road map example:

Re-introduce the costs of paths in the NET



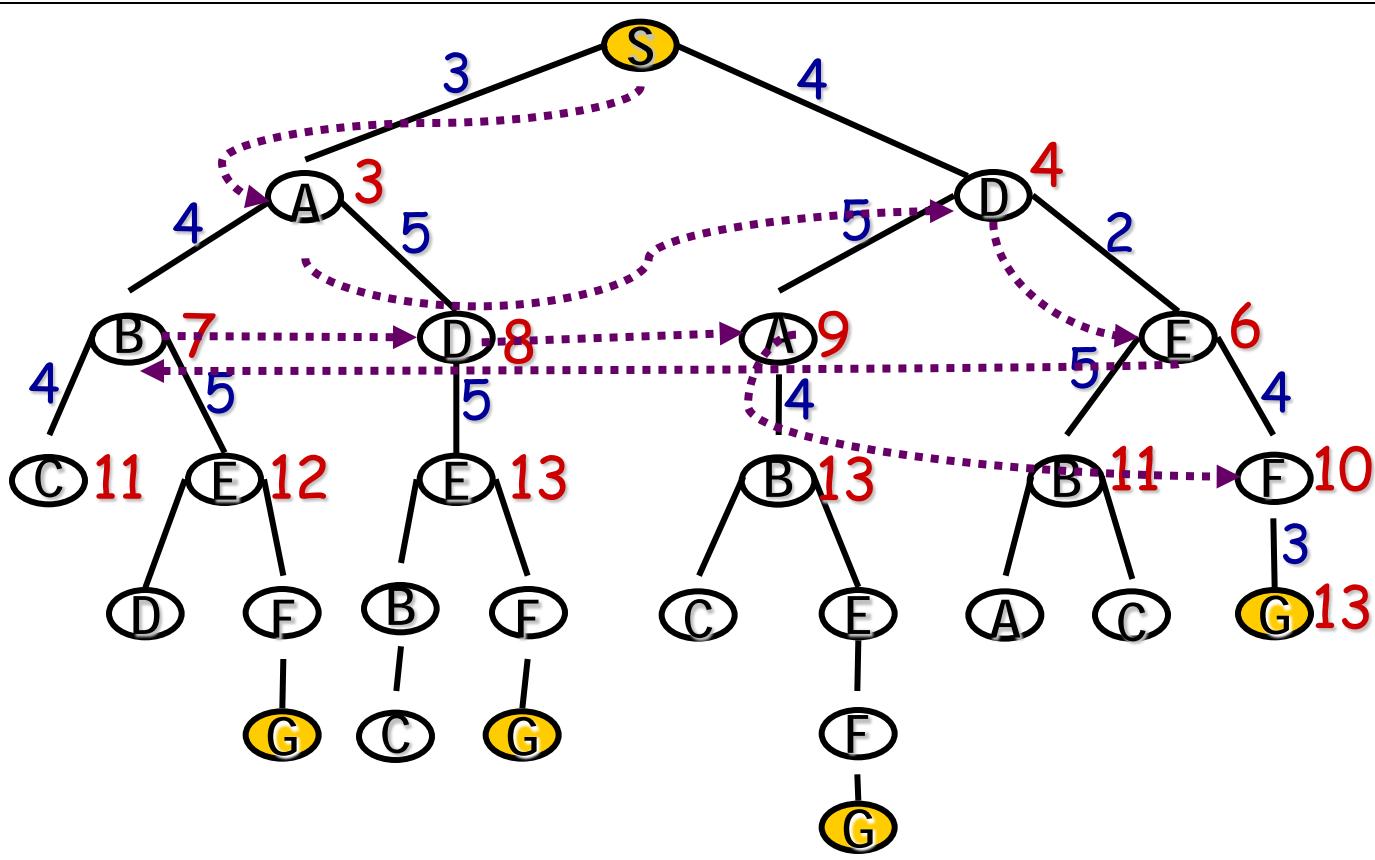
A look at Uniform cost search

= uniformed best-first

- The Uniform Cost Search is also referred to as Cheapest-First Search best because it is guaranteed to find the path with the cheapest total cost.
- It is also referred to as **Dijkstra's** single-source shortest algorithm
- The Uniform Cost Search is related to the Breadth-First Search
- It determines the node to be expanded next by calculating the cost so far to reach each node in the frontier from the root and picks the path with the lowest total cost

A look at Uniform cost search

= uniformed best-first



□ At each step, select the node with the lowest accumulated cost.

□ with $g(n)$ = the sum of edge costs from start to n

Now incorporate heuristic estimates

- Replace the ‘accumulated cost’ in the ‘Uniform cost search’ by a function:

$$f(\text{path}) = \text{cost}(\text{path}) + h(\text{endpoint_path})$$

F(n)

g(n)

h(n)

- where:

$\text{cost}(\text{path})$ = the accumulated cost of the partial path

$h(n)$ = a heuristic estimate of the cost remaining
from n to a goal node

$f(\text{path})$ = an estimate of the cost of a path extending
the current path to reach a goal

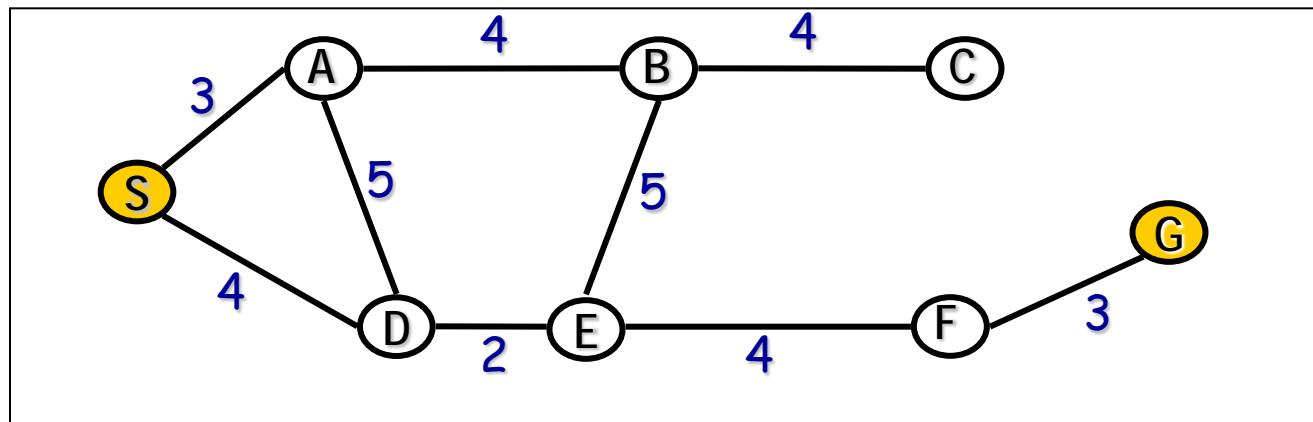
$$F(n) = g(n) + h(n)$$

Heuristic Searches - A* Algorithm

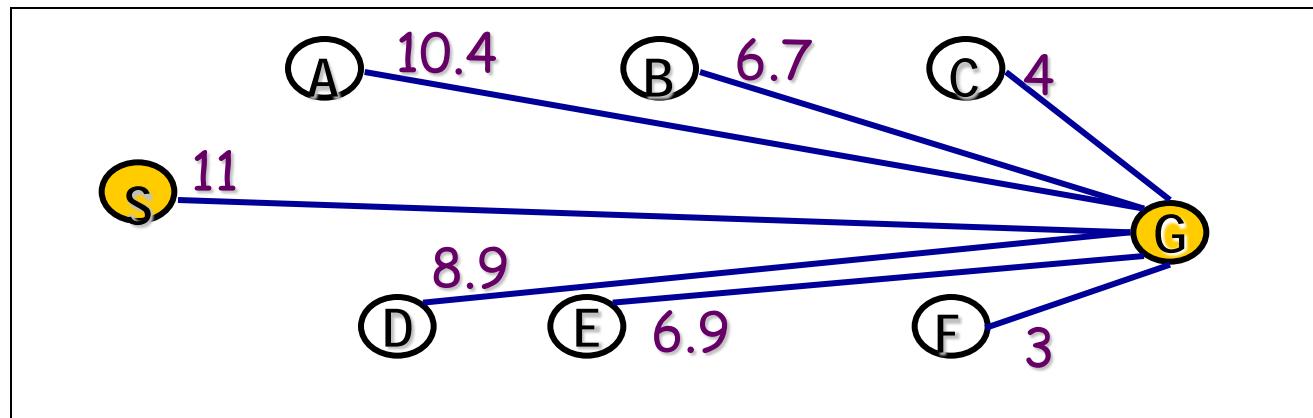
- Combines the cost so far and the heuristic estimated cost to the goal. That is $f(n) = g(n) + h(n)$
This gives us estimated cost of the cheapest solution through path n
- It can be proved to be **optimal** and **complete** providing that the heuristic is **admissible**
 - That is the heuristic must never over estimate the cost to reach the goal
- But, the number of nodes that have to be searched still grows exponentially

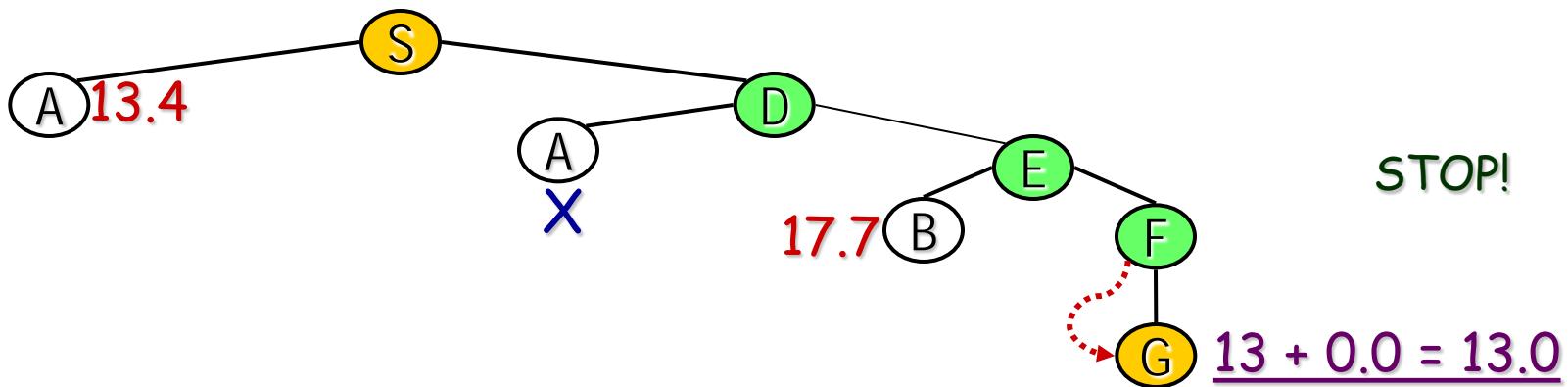
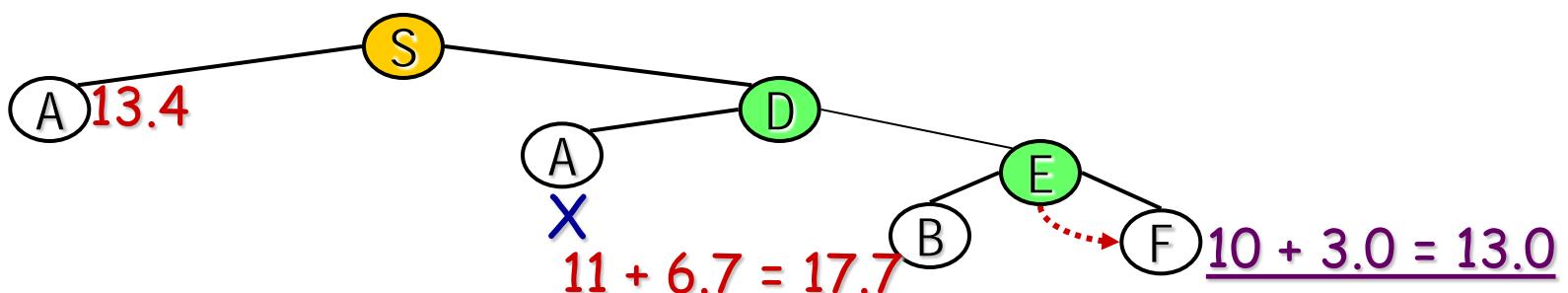
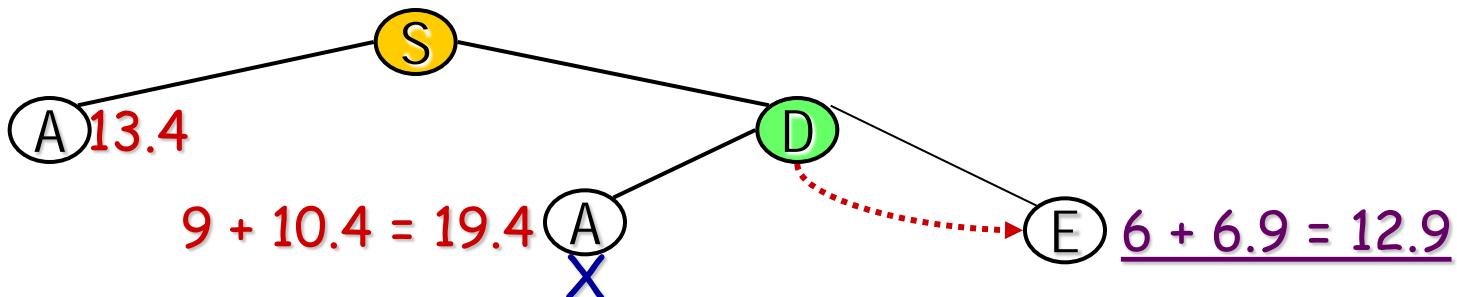
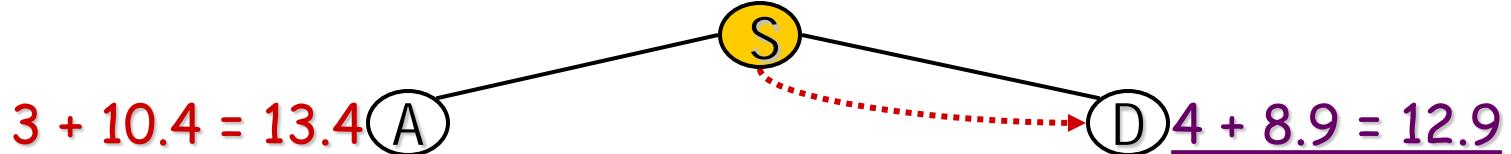
Example: road map

- Imagine the problem of finding a route on a road map. The paths distances between nodes define $g(n)$:

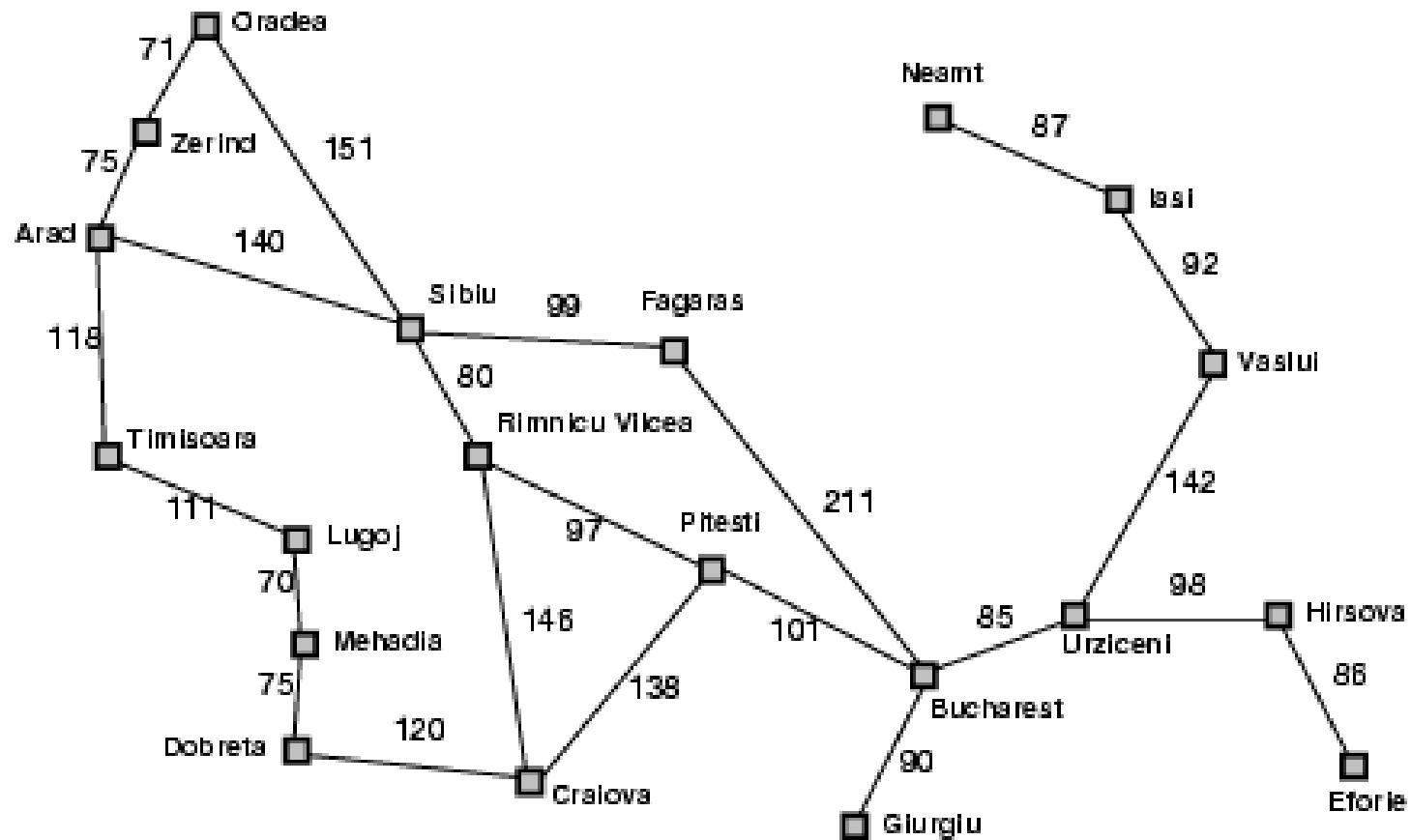


- Define $h(n) =$ the straight-line distance from node to G

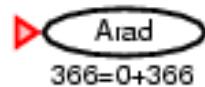




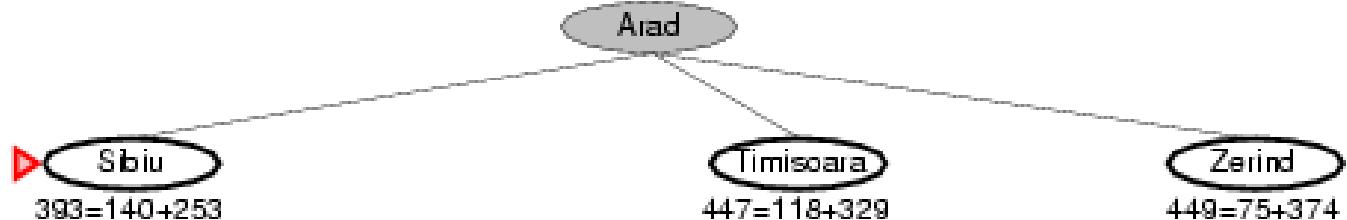
Example: A* algorithm for Romania Tour with step costs in km



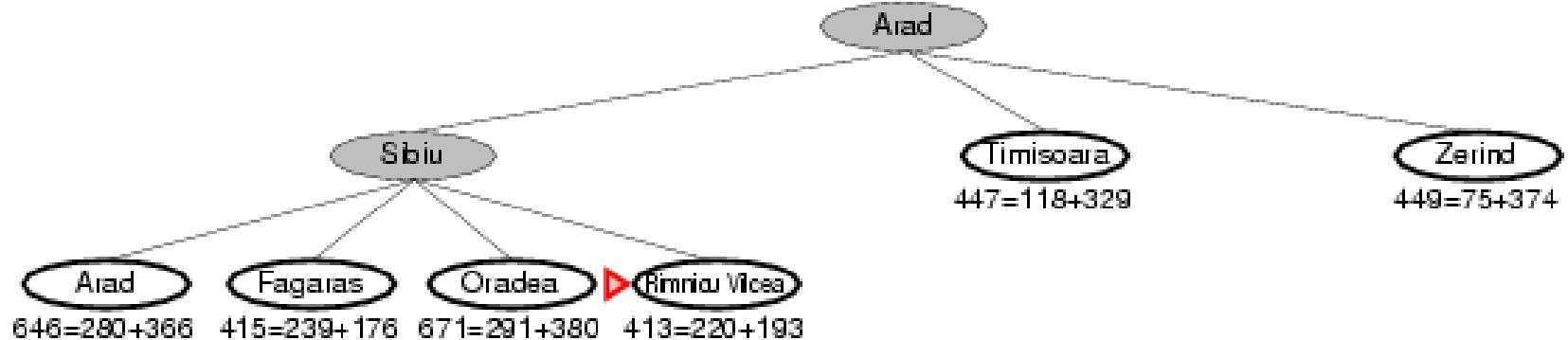
Example: A* Algorithm for Romania Tour



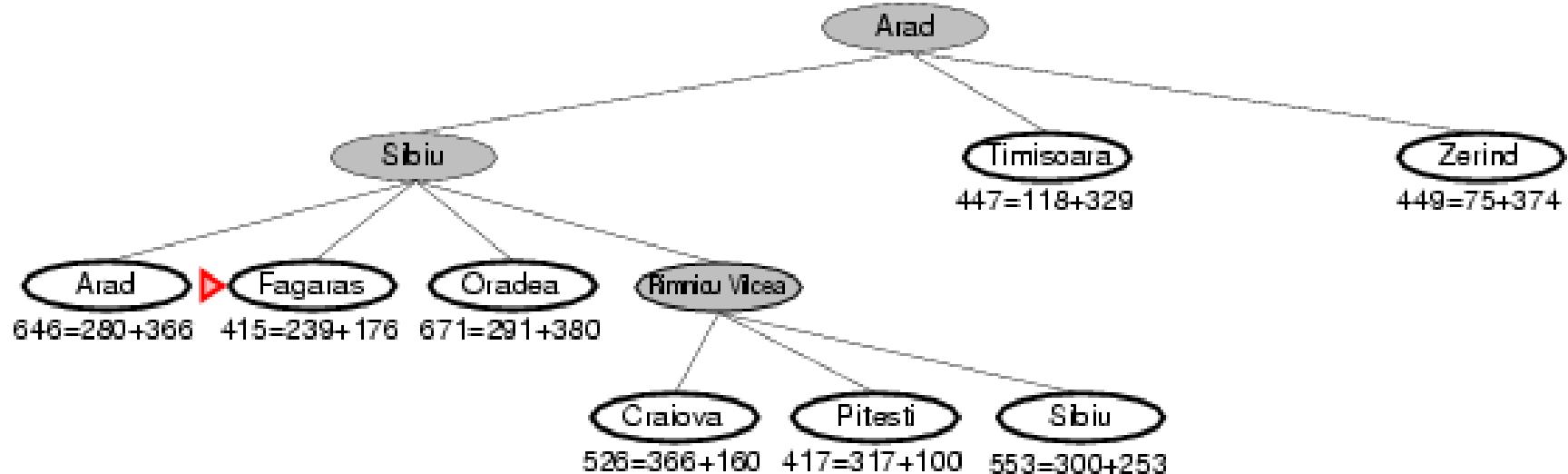
Example cont...



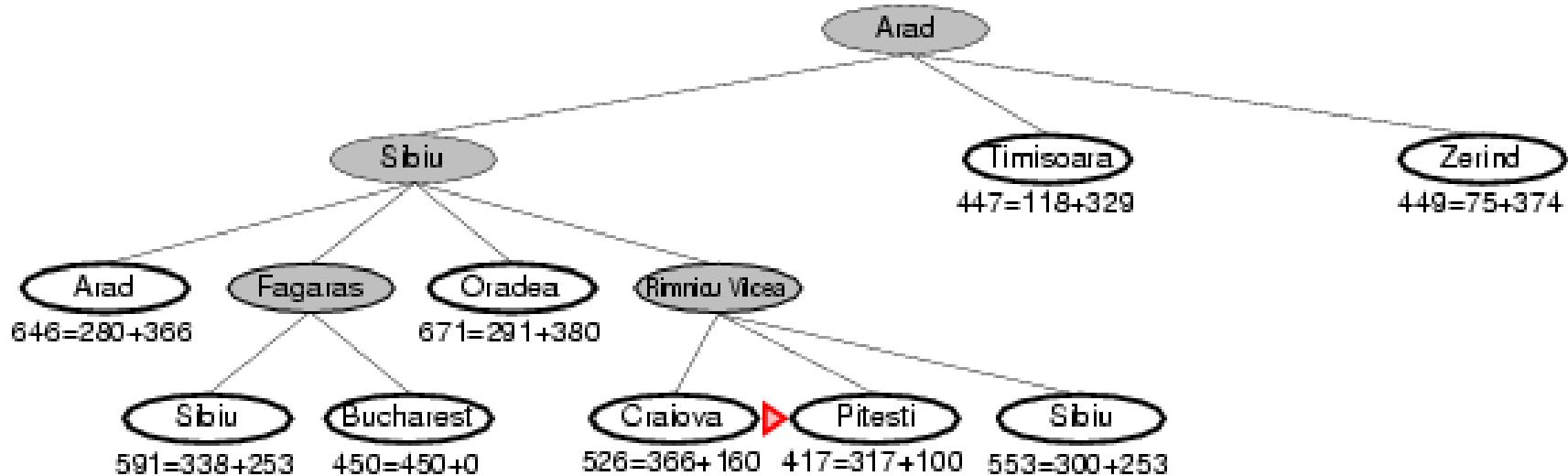
Example cont...



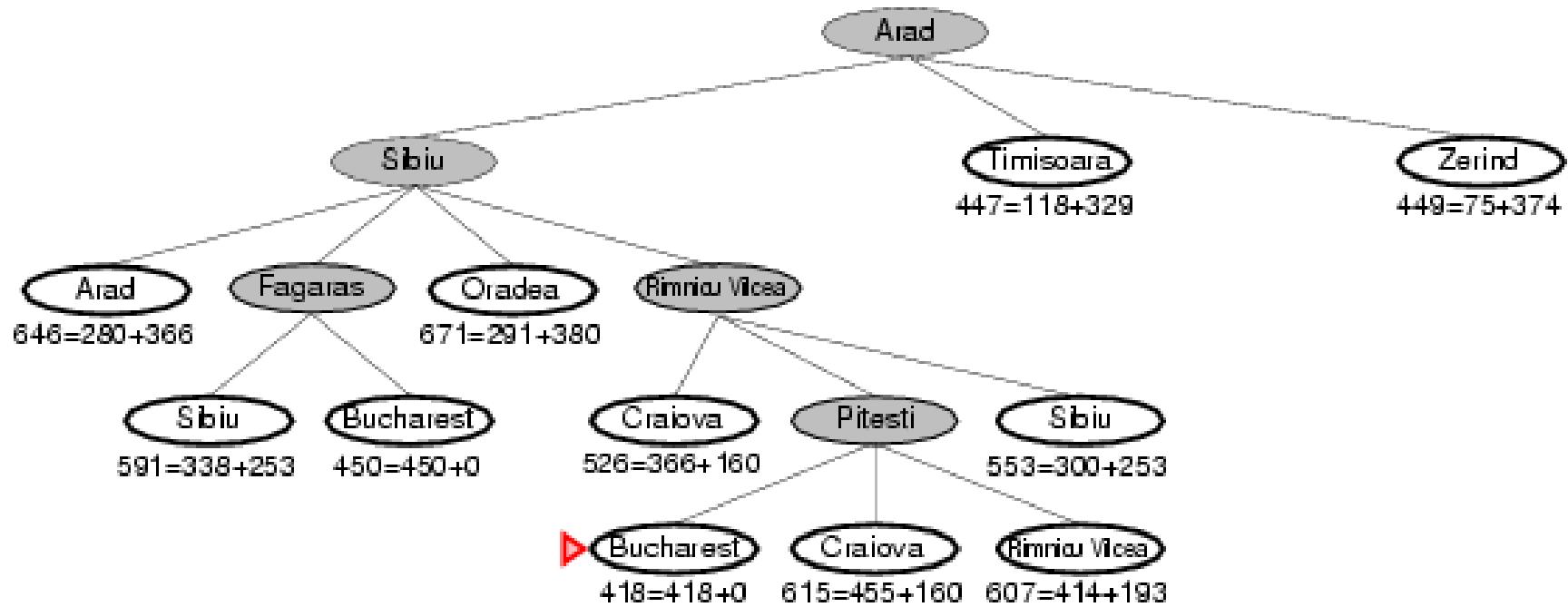
Example cont...



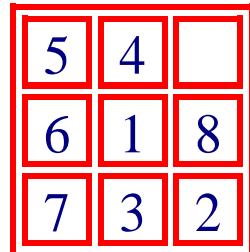
Example cont...



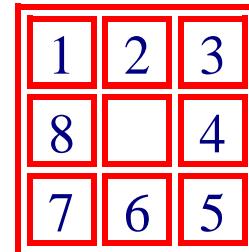
Example cont...



Example: A* Algorithm for 8-Puzzle



Initial State



Goal State



- Typical solution is about twenty steps
- Branching factor is approximately three. Therefore a complete search would need to search 3^{20} states. But by keeping track of repeated states we would only need to search $9!$ ($362,880$) states
- But even this is a lot (imagine having all these in memory)
- Our aim is to develop a heuristic that does not over estimate (it is admissible) so that we can use A* to find the optimal solution

Heuristic Searches - Possible Heuristics



- H_1 = the number of tiles that are in the wrong position ($=7$)
- H_2 = the sum of the distances of the tiles from their goal positions using the Manhattan Distance ($=18$)

Both are admissible but which one is best?

Test from 100 runs with varying solution depths

Depth	Search Cost		
	IDS	A*(h ₁)	A*(h ₂)
2	10	6	6
4	112	13	12
6	680	20	18
8	6384	39	25
10	47127	93	39
12	364404	227	73
14	3473941	539	113
16		1301	211
18		3056	363
20		7276	676
22		18094	1219
24		39135	1641

Number of nodes expanded

H₂ looks better as fewer nodes are expanded. But why?

Effective Branching Factor

Search Cost				EBF			
Depth	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$	
2	10	6	6	2.45	1.79	1.79	
4	112	13	12	2.87	1.48	1.45	
6	680	20	18	2.73	1.34	1.30	
8	6384	39	25	2.80	1.33	1.24	
10	47127	93	39	2.79	1.38	1.22	
12	364404	227	73	2.78	1.42	1.24	
14	3473941	539	113	2.83	1.44	1.23	

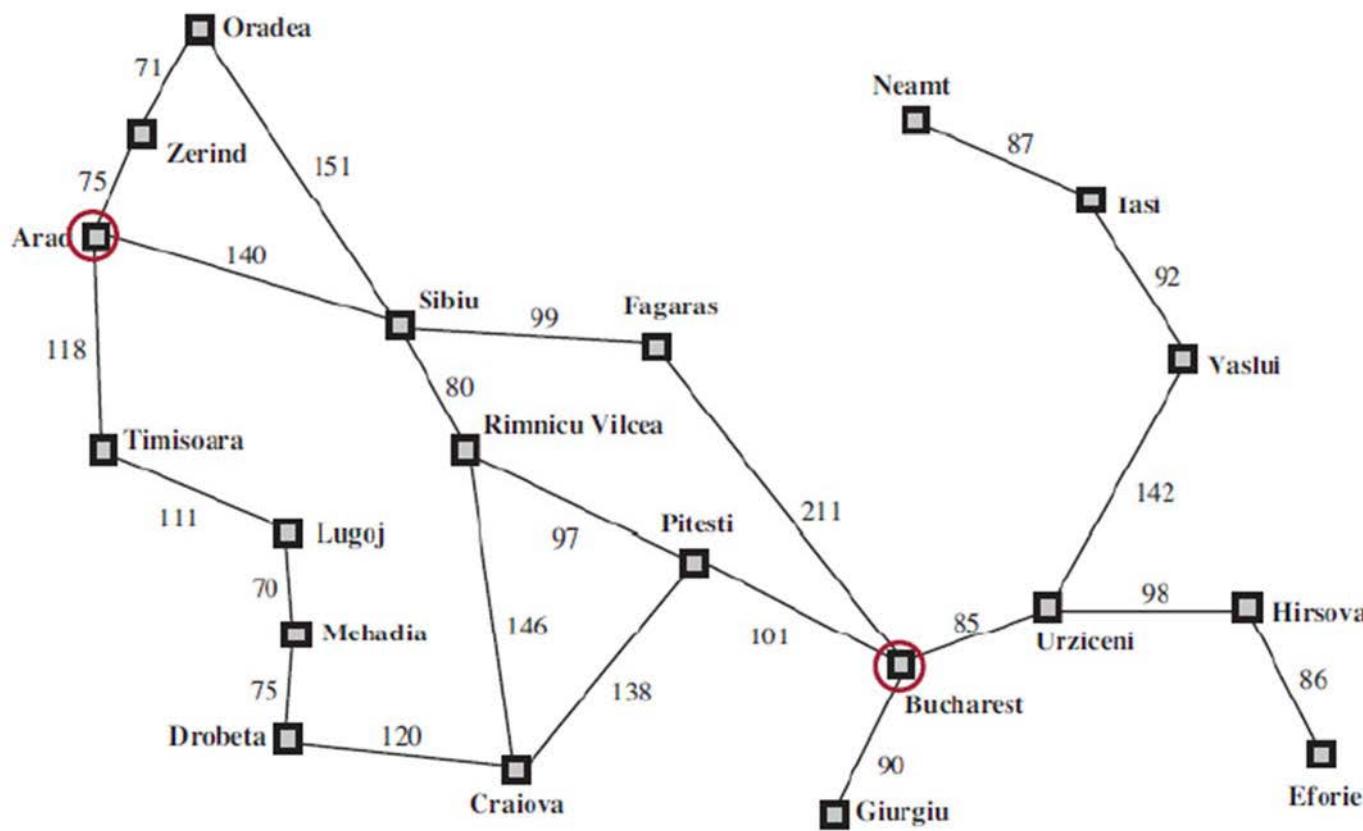
- H_2 has a lower branching factor and so fewer nodes are expanded
- Therefore, one way to measure the quality of a heuristic is to find its average branching factor
- H_2 has a lower EBF and is therefore the better heuristic

Summary

- Blind Search is very expensive
- A* Search with info (heuristics) is far better
- Info can be difficult to incorporate
- The more info, the better

Example Problem: Romania Tour

- On holiday in Romania; currently in Arad, will leave tomorrow for Bucharest
- Formulate goal:
 - be in Bucharest
- Formulate problem:
 - **states:** various cities
 - **actions:** drive between cities
- Find solution:
 - sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

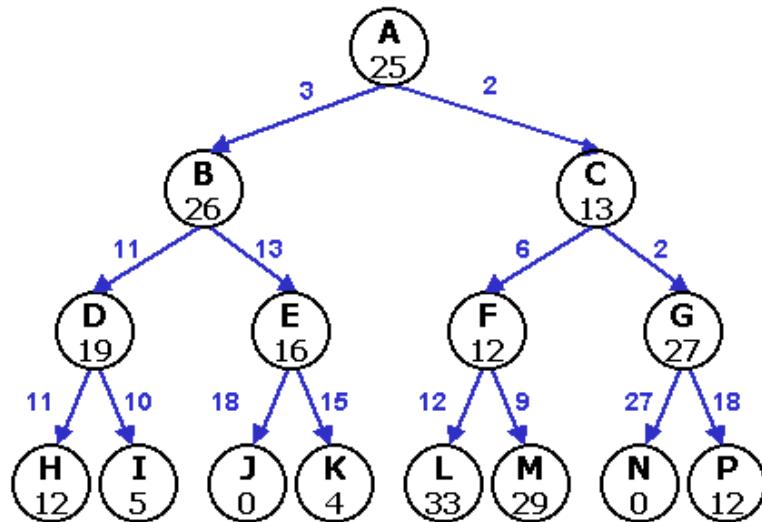


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Refer to Russell & Norvig Chapter 3

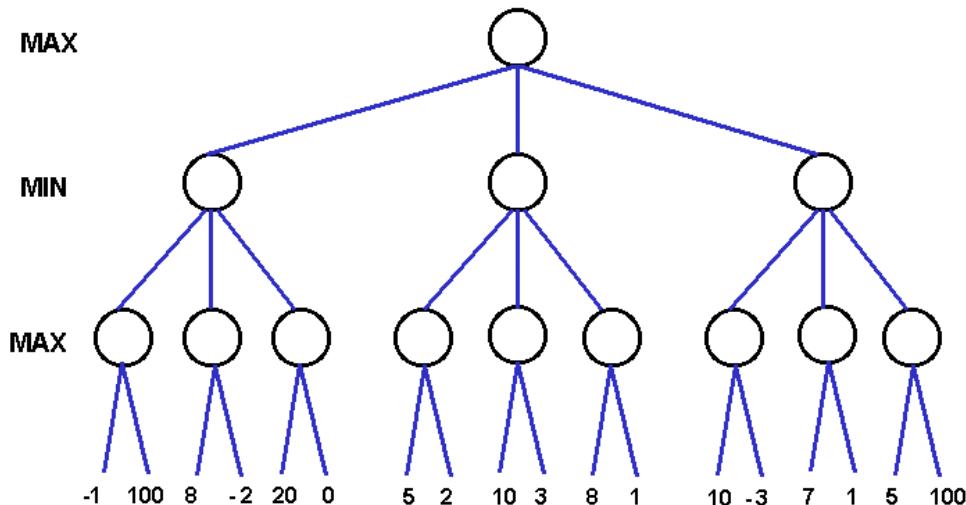
Search Exercise

- (a) A search tree is shown below where each circle represents a node corresponding to a state in the search space. The estimated cost (i.e. h function) for finding a solution from a node is shown in its circle. The two nodes with $h = 0$ are goal states and the other terminal nodes are dead-ends. Actual link costs are marked on the links between the nodes. Thus the path cost (i.e. g function) of a node is equal to the sum of the link costs from the root to that node.



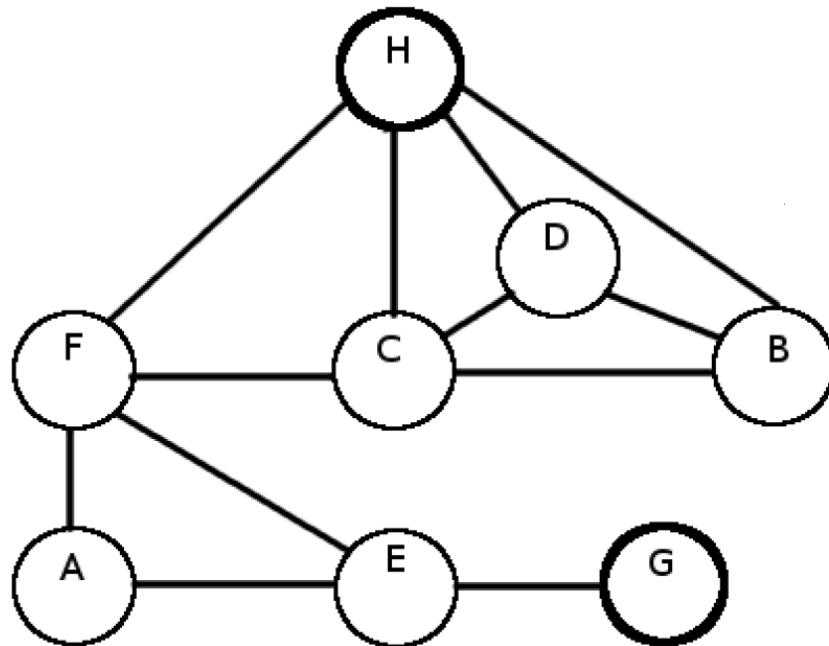
Using the greedy search algorithm, give the sequence of nodes expanded before a goal state is reached. What is the solution path and what is its path cost?

- (b) Consider the MIN-MAX game tree given below.



- i) Fill in the utility function values at each node [the blank circles] in the MIN-MAX tree above, and mark the first move path, from the root node with a thick line.

- ii) Cross out the branches that are pruned by α - β pruning. How many nodes did you not have to visit with α - β pruning when compared to the full MIN-MAX search above? Show all intermediate values at each node as they get updated.
- (c) An alien has changed his name to “Ford Prefect” when he came to Earth because he thought he would blend in with the dominant life forms. Ford’s friend needs a new name too. Ford suggests he start with “Ford Mustang”. The friend decides to use his 6.034 search skills to navigate through a sea of choices to find a good name. The friend is now faced with the graph shown below

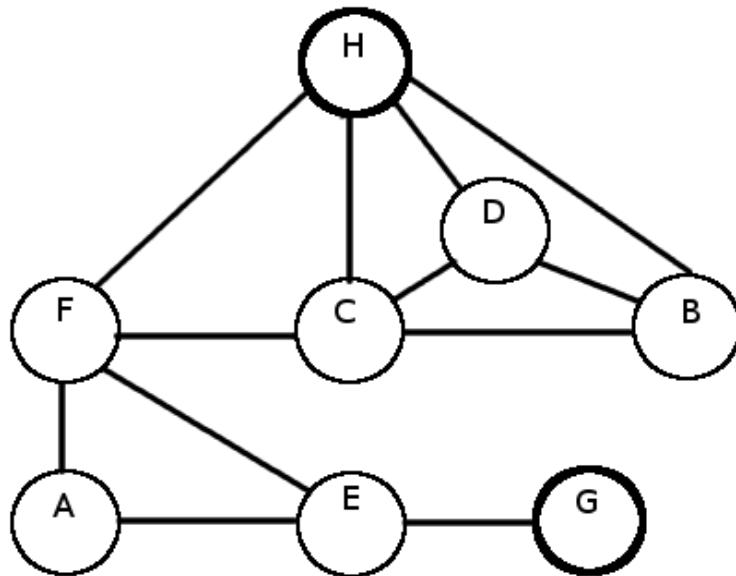


Note: In all search problems, use alphabetical order to break ties when deciding the priority to use for extending nodes.

- Starting at node **H**, find the Depth-First Search (DFS) path to **G**. Draw the DFS tree to show how you arrive at this solution
- How many times do you encounter loops during your search? State all paths in your search that contain loops

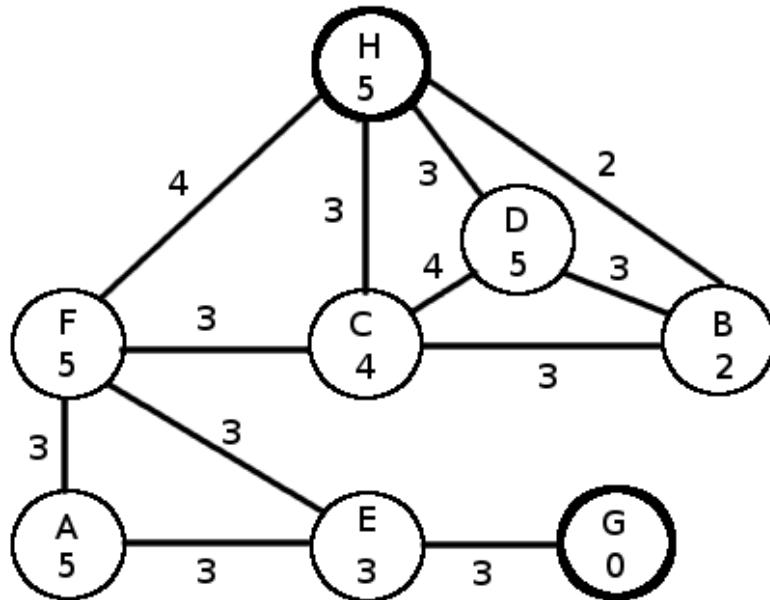
(d) Use the depth-first search to find ALL possible paths from the root node **H** to the goal node **G**

- Draw the tree for this search
- List ALL possible paths

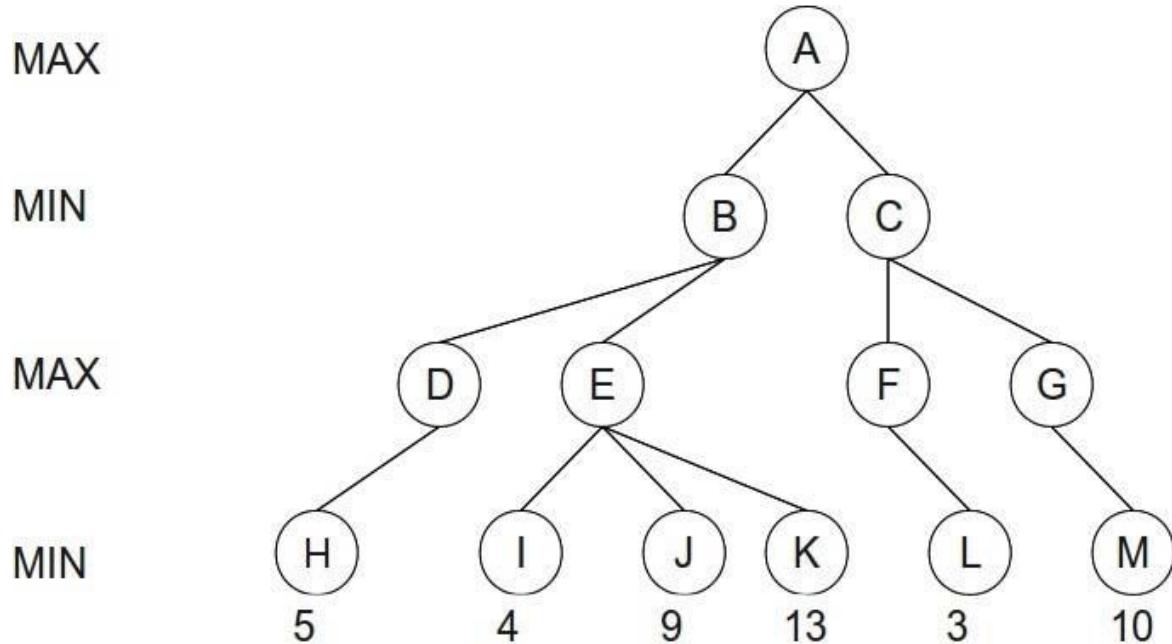


(e) Use the A* search to find the path from the root node to the goal node. Straight-line distances from each node to the goal node are indicated within the nodes

- Draw the corresponding tree and show your work
- Path from root to goal



(f) Consider the following tree.



- i. Perform Minimax on this tree. Write the minimax value associated with each node inside the node, next to its corresponding node letter. Indicate the direction/move that the agent should take
- ii. Perform Alpha Beta search on this tree. Indicate pruning by striking through the appropriate edge(s). Mark your steps by indicating both the temporary and final alpha and beta values next to each node

COMP 308

ARTIFICIAL INTELLIGENCE

PART 3.1 – PROBLEM SOLVING

- SEARCHING

Njeri Ireri

Jan – April 2020

We Shall Discuss

- Introduction to problem solving
- Problem Solving Techniques
- Search as problem solving technique
- Problem Definition
- Search Terminology
- Evaluating a search

Workers are always Searching



Brief info on Ants, <http://www.winnipeg-bugline.com/ants.html>,
<http://www.boston.com/globe/magazine/2002/0623/frontiers.htm>

Problem Solving Techniques in A.I

- Broad Approaches
 - using search techniques
 - e.g. in Games
 - modeling
 - using Knowledge Base Systems (KBS)
 - using Machine Learning techniques e.g. Artificial Neural Networks, Decision Trees, Case-base reasoning, Genetic algorithms, ..

Searching as a Problem Solving Technique

- **Searching** is the process of looking for the solution of a problem through a set of possibilities (state space)
- **Search conditions** include:
 - Current state – where one is;
 - Goal state – the solution reached; check whether it has been reached;
 - Cost of obtaining the solution
- The **solution** is a path from the current state to the goal state

Searching as a Problem Solving Technique

Process of Searching

- Searching proceeds as follows:
 - Check the current state;
 - Execute allowable actions to move to the next state;
 - Check if the new state is the solution state; if it is not, then the new state becomes the current state and the process is repeated until a solution is found or the state space is exhausted

Search Problem

- The **search problem** consists of finding a **solution plan**, which is a path from the current state to the goal state
- **Representing search problems**
 - A search problem is represented using a directed graph (tree)
 - The states are represented as nodes while the allowed steps or actions are represented as arcs (branches)
- **A search problem is defined by specifying:**
 - State space;
 - Start node;
 - Goal condition, and a test to check whether the goal condition is met;
 - Rules giving how to change states
 - Path cost

Problem Definition - Example, 8 puzzle

5	4	
6	1	8
7	3	2

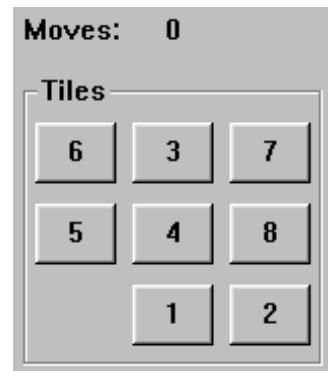
Initial State

1	4	7
2	5	8
3	6	

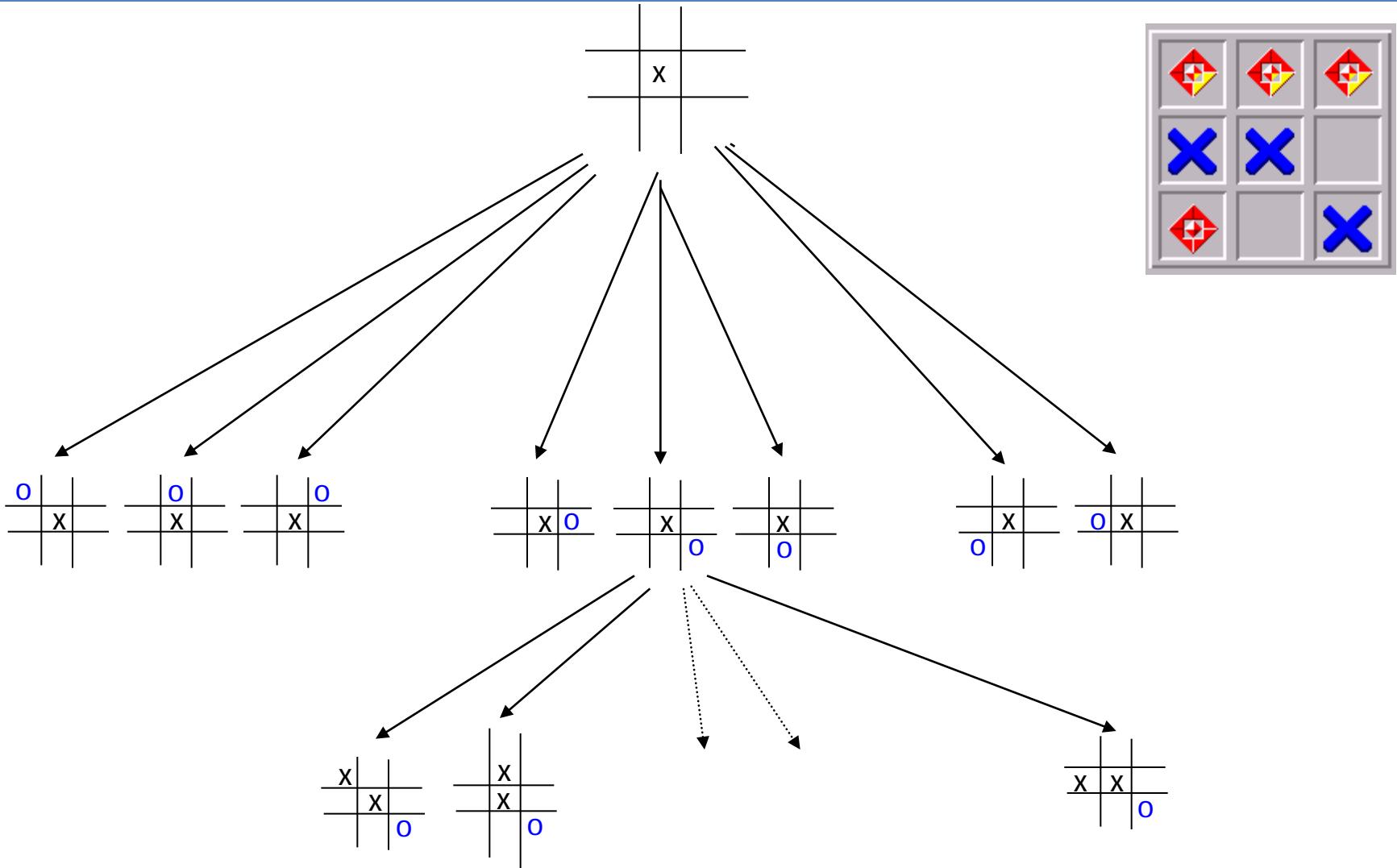
Goal State

Problem Definition - Example, 8 puzzle

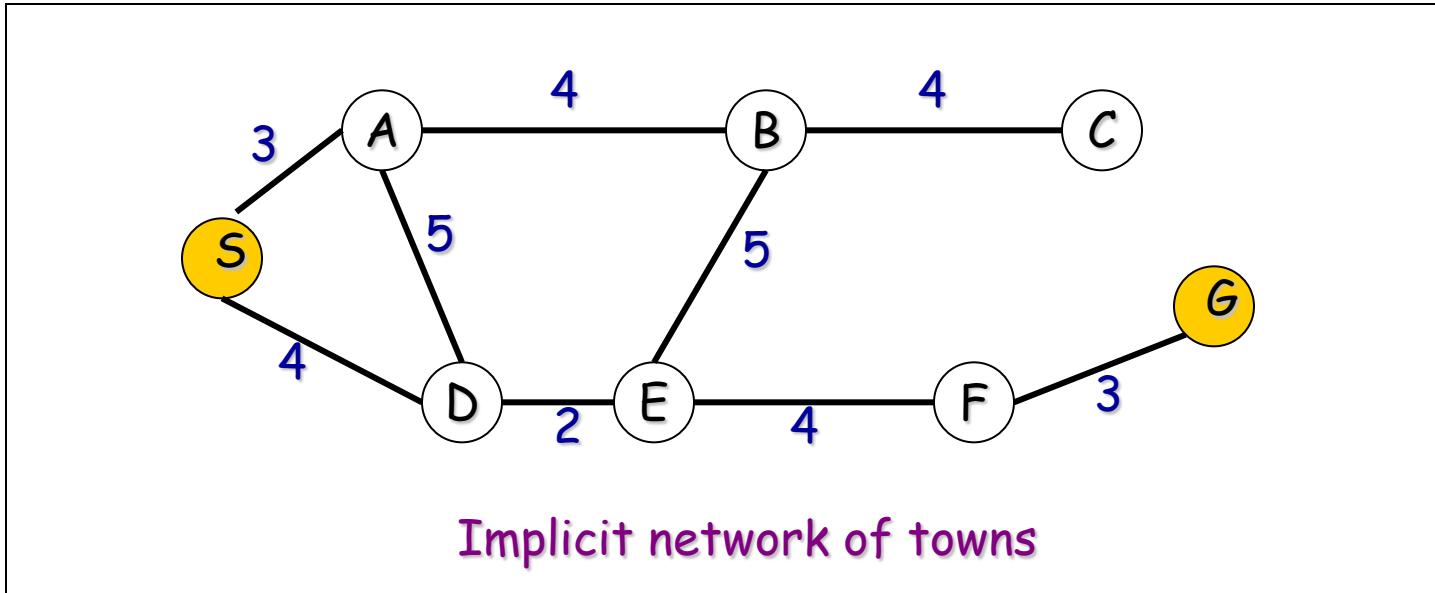
- States
 - ▣ A description of each of the eight tiles in each location that it can occupy. It is also useful to include the blank
- Operators/Action
 - ▣ The blank moves left, right, up or down
- Goal Test
 - ▣ The current state matches a certain state (e.g. one of the ones shown on previous slide)
- Path Cost
 - ▣ Each move of the blank costs 1



Problem Definition - Example, tic-tac-toe

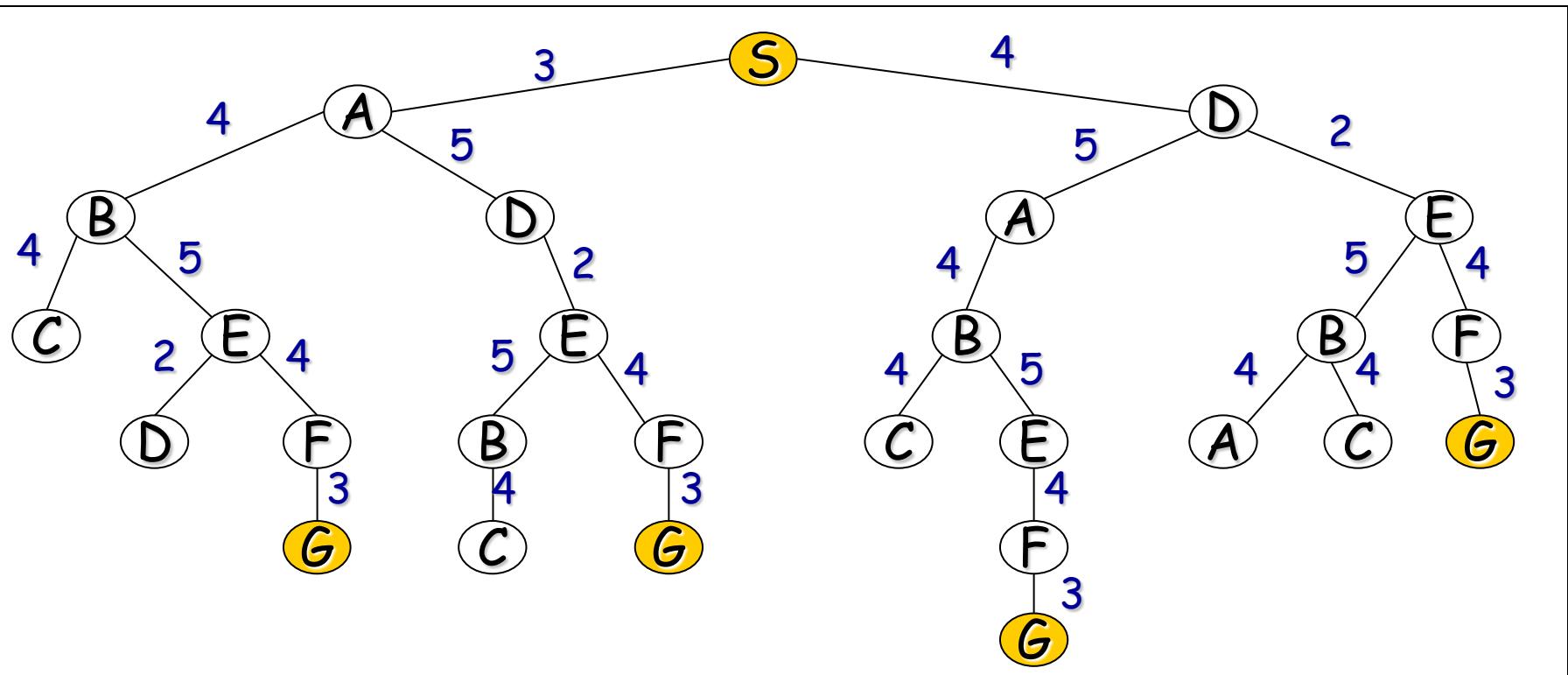
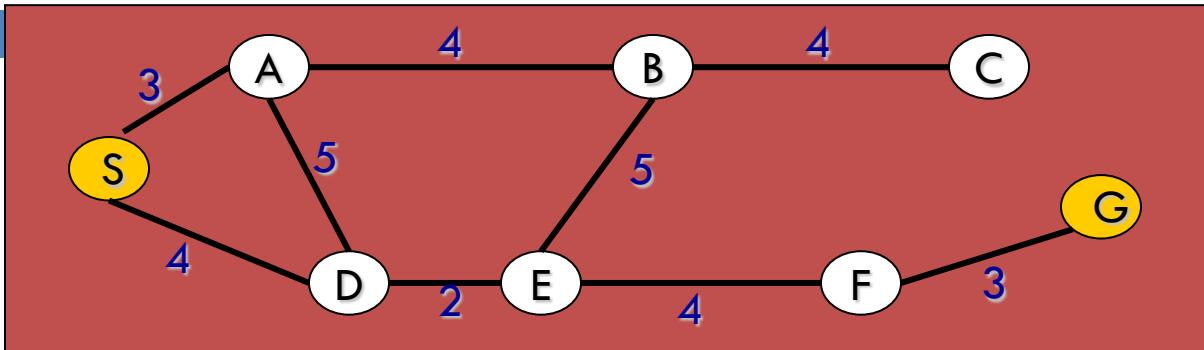


Tree/Path Example:



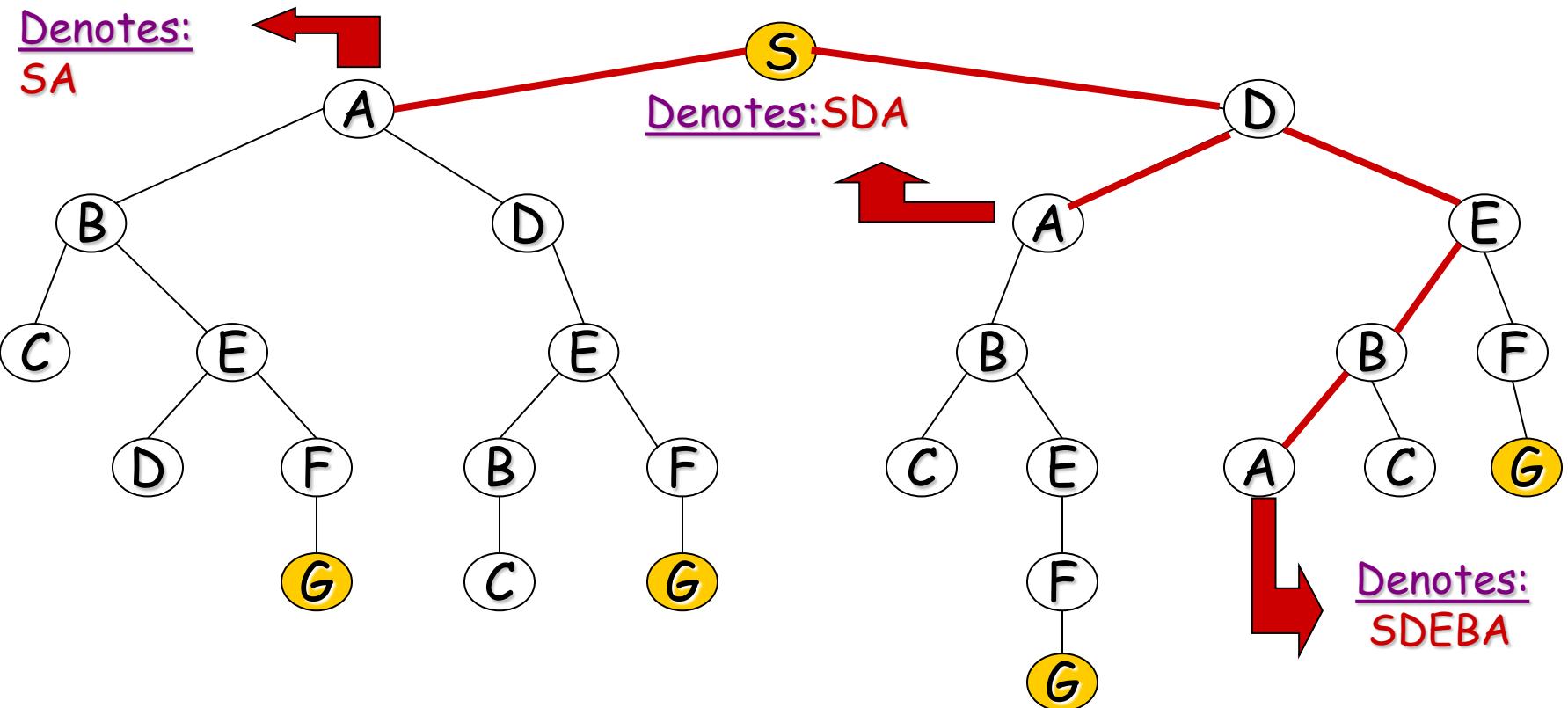
- Two possible tasks:
 - 1. FIND a (the) path. = computational cost
 - 2. TRAVERSE the path. = travel cost
- 2. relates to finding optimal paths

The associated loop-free tree of partial paths



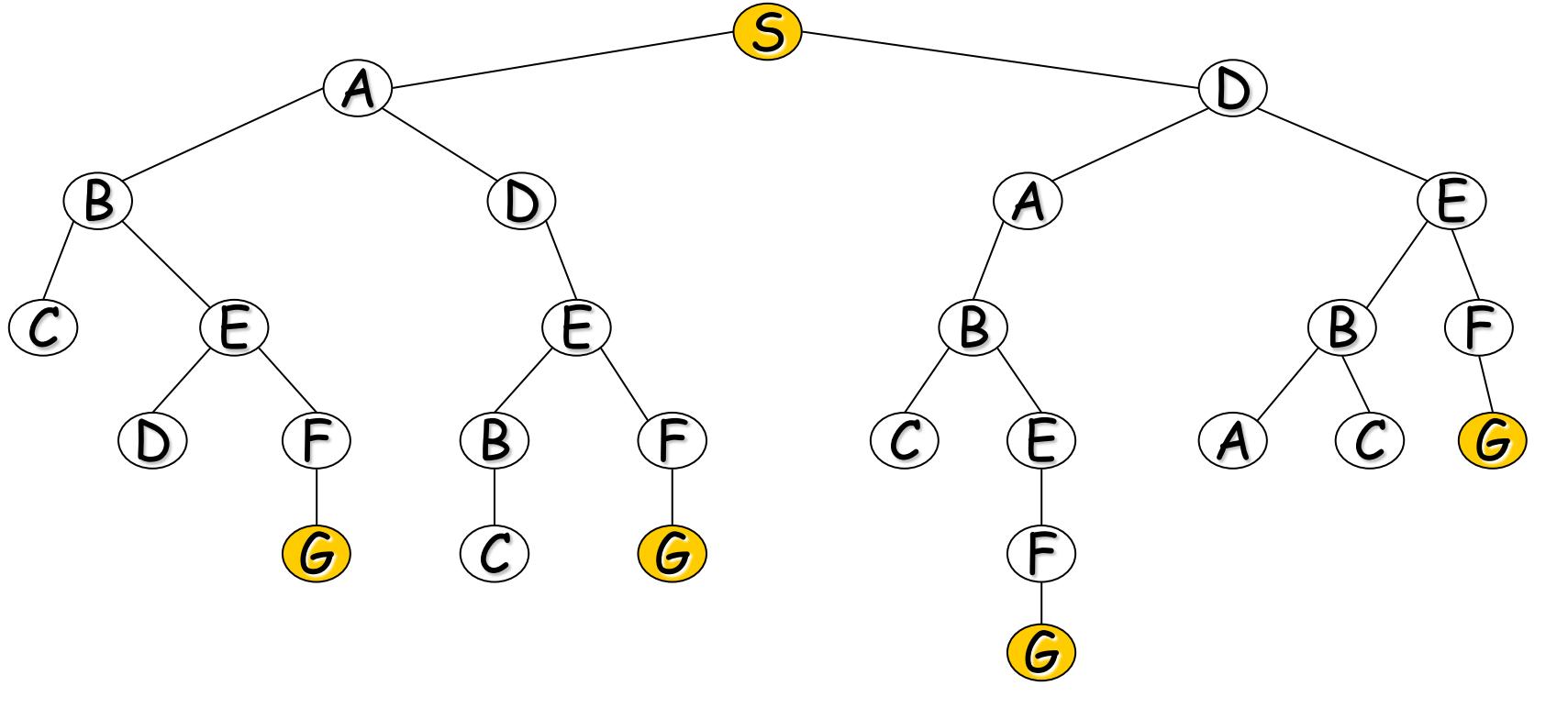
Paths:

- We are not interested in optimal paths here, so we can drop the costs



Note: Nodes do not denote themselves, but denote the partial path from the root to themselves!!

Terminology:



- Node, link (or edge), branch, arc
- Parent, child, ancestor, descendant
- Root node, goal node
- Expand / Open node / Closed node / Branching factor

Using a Tree – The Obvious Solution?

- **But**
 - It can be wasteful on space
 - It can be difficult to implement, particularly if there are varying number of children (as in tic-tac-toe)
 - It is not always obvious which node to expand next
 - We may have to search the tree looking for the best **leaf node** (sometimes called the **fringe** or **frontier** nodes). This can obviously be computationally expensive

How Good is a Solution?

- Does our search method actually find a solution?
- Is it a good solution?
 - Path Cost
 - Search Cost (Time and Memory)
- Does it find the optimal solution?
 - But what is optimal?

Evaluating a Search

- **Completeness**
 - ▣ Is the strategy guaranteed to find a solution?
- **Time Complexity**
 - ▣ How long does it take to find a solution?
- **Space Complexity**
 - ▣ How much memory does it take to perform the search?
- **Optimality**
 - ▣ Does the strategy find the optimal solution where there are several solutions?

Search Trees

- Some issues:
 - Search trees grow very quickly
 - The size of the search tree is governed by the branching factor
 - Even this simple game tic-tac-toe has a complete search tree of 984,410 potential nodes
 - The search tree for chess has a branching factor of about 35

Exercise

1. How are problems solved in artificial intelligence?
2. What is searching?
3.
 - (a) What are the things that could specify a search problem?
 - (b) Supposing you had a robot that is supposed to maneuver it self on a factory floor cluttered with numerous machines and boxes containing both raw and finished materials from the back to the front of the factory. What would be specified for the case in (a)

Exercise

4. (a) Playing the 8 Puzzle game, draw a search tree to level three for the initial game state given in figure 1
(b) How many moves would you require to complete the game given in figure 1

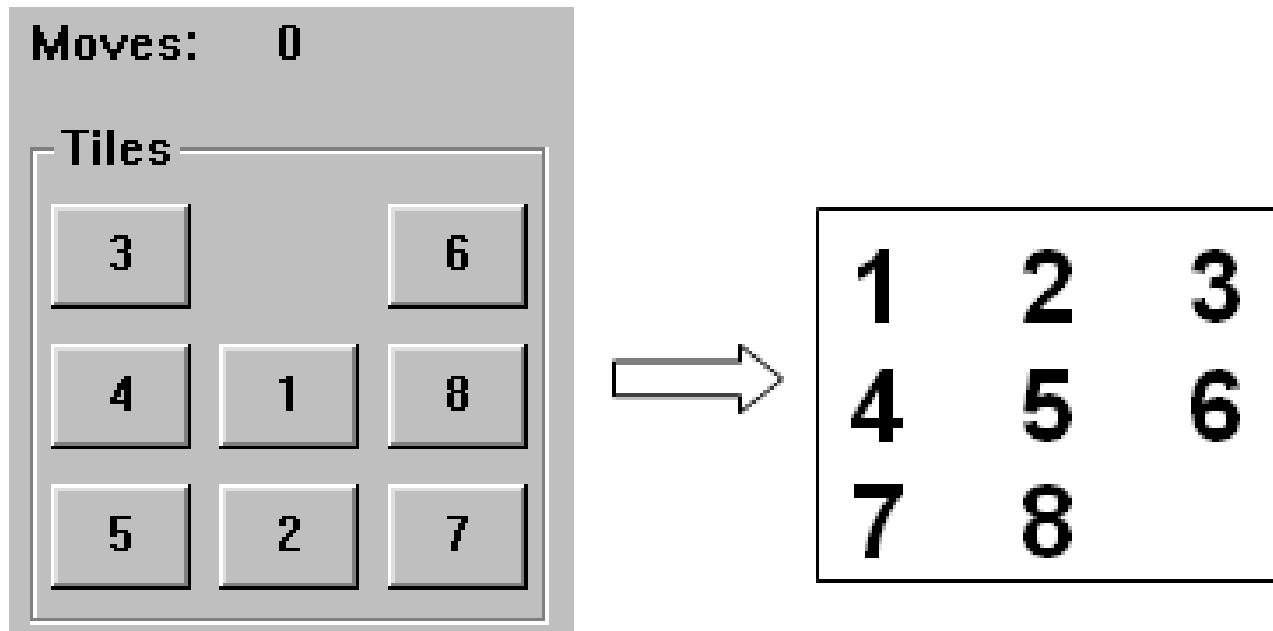


Figure 1. 8 Puzzle (Left-Start State and Right-Goal State)

COMP 308

ARTIFICIAL INTELLIGENCE

PART 3.2 – UNINFORMED (BLIND) SEARCH

Njeri Ireri

Jan – April 2020

We Shall Discuss

- What is Uninformed Search?
- Uninformed Search Methods
 - Depth-first search
 - Breadth-first search
 - Non-deterministic search
 - Iterative deepening search
 - Bi-directional search

Uninformed Search?

- Simply searches the state space (or NET)
- Can only distinguish between goal state and non-goal state
- Sometimes called **Blind search** as it has no information or knowledge about its domain

Uninformed Search Characteristics

- Blind Searches have **no preference** as to which state (node) that is expanded next
- The different **types** of blind searches are **characterised by the order** in which **they expand** the nodes
 - This can have a dramatic effect on how well the search performs when measured against the four criteria we defined in an earlier lecture
 - Search evaluation criteria - Completeness, Time Complexity, Space Complexity, Optimality (of given solution when there are several solutions to choose from)

Uninformed (Blind) Search Methods

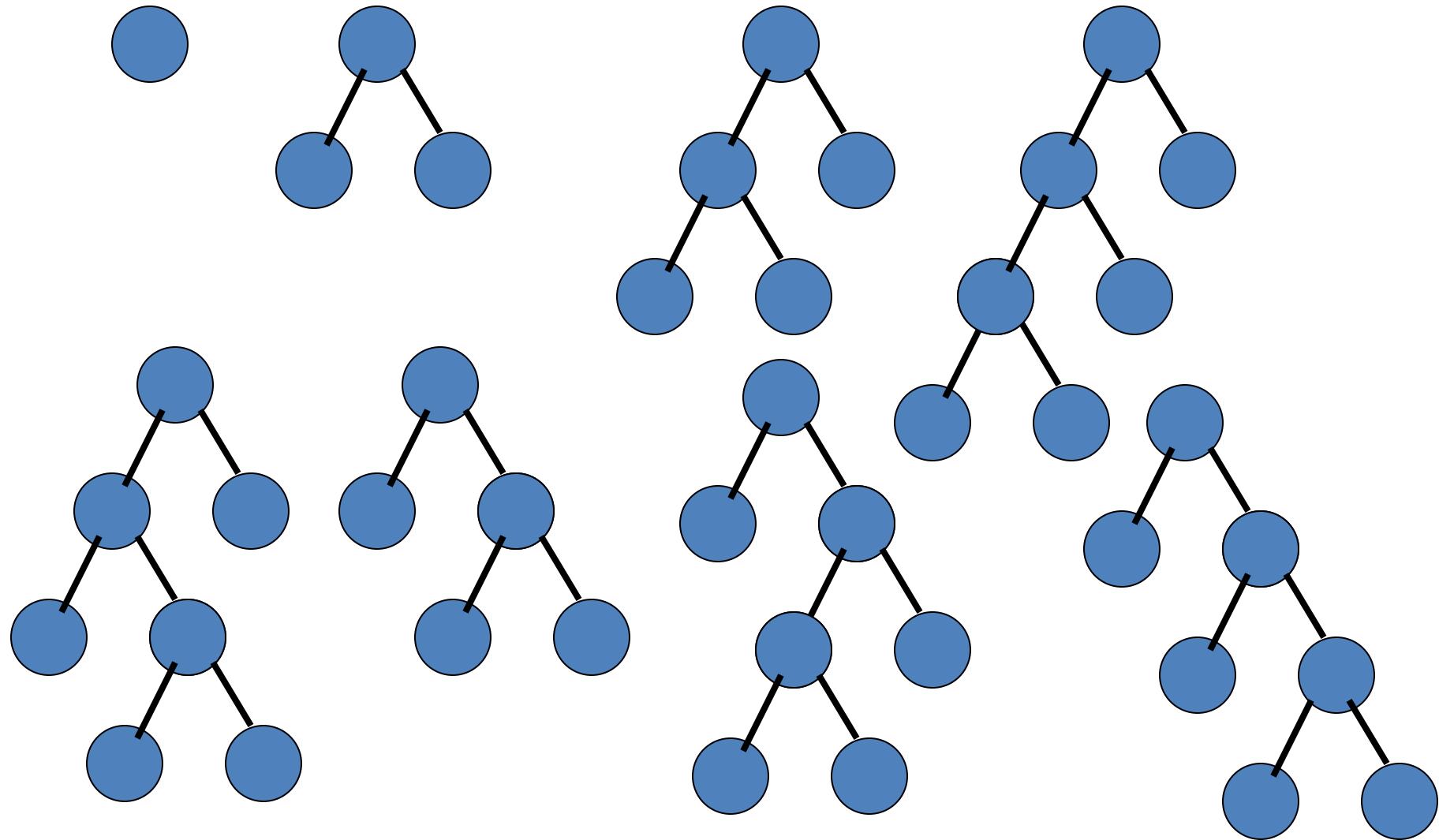
- Methods that do not use any specific knowledge about the problem
- These are:
 - ▣ Depth-first search
 - ▣ Breadth-first search
 - ▣ Non deterministic search
 - ▣ Iterative deepening search
 - ▣ Bi-directional search

1. Depth-first Search



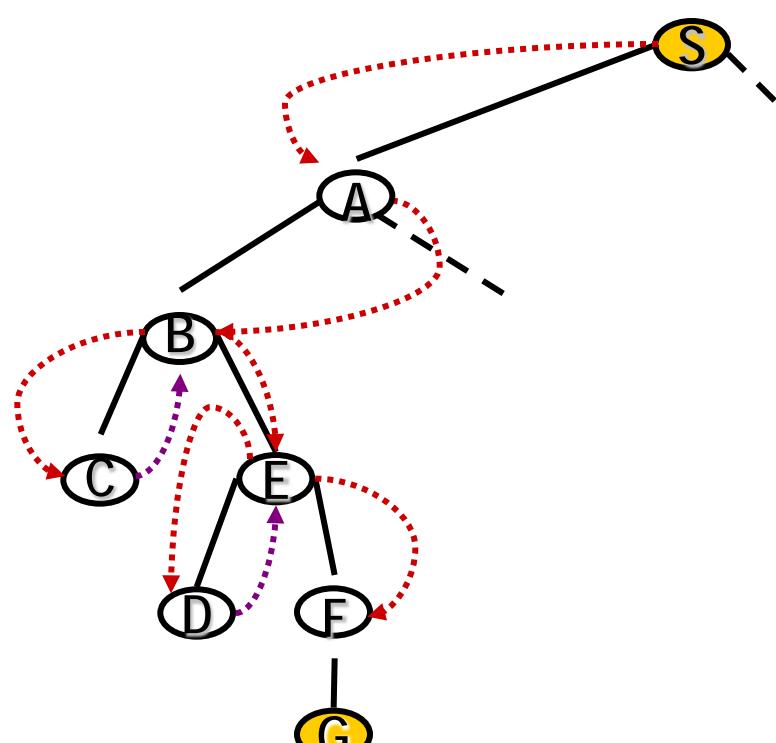
- Expand the tree as deep as possible, returning to upper levels when needed

Depth-First Search



Depth-First Search

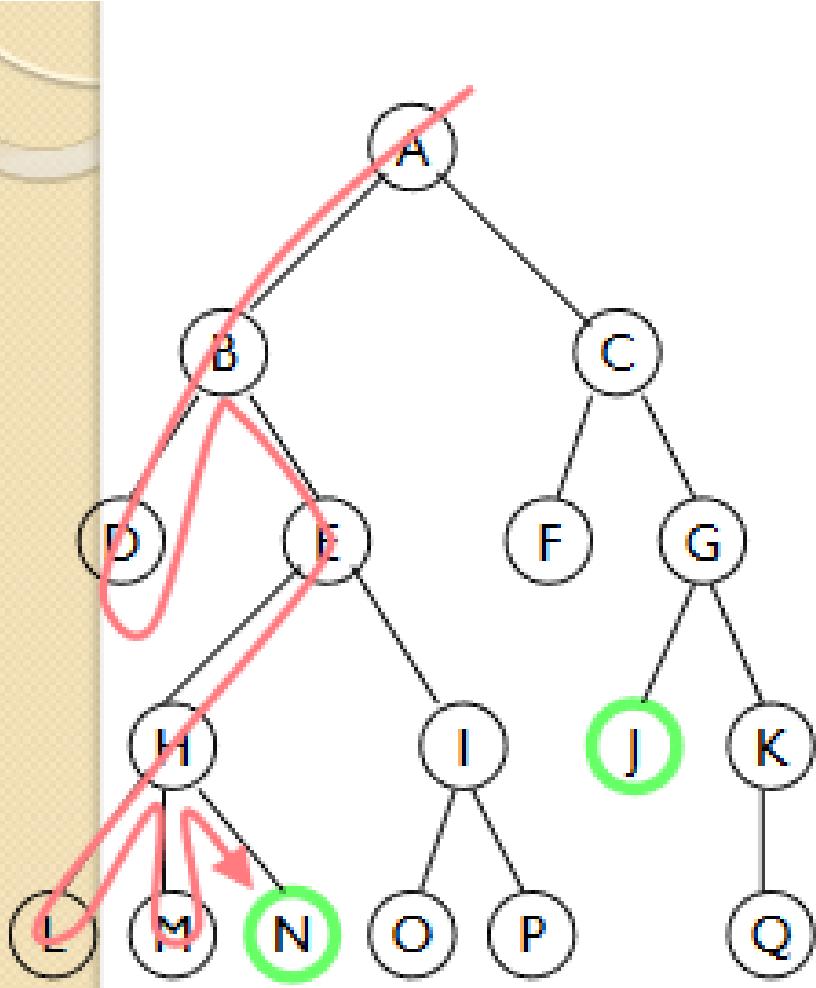
= Chronological backtracking



- Select a child
 - convention: left-to-right
- Repeatedly go to next child, as long as possible
- Return to left-over alternatives (higher-up) only when needed

Depth-First Search

= Chronological backtracking



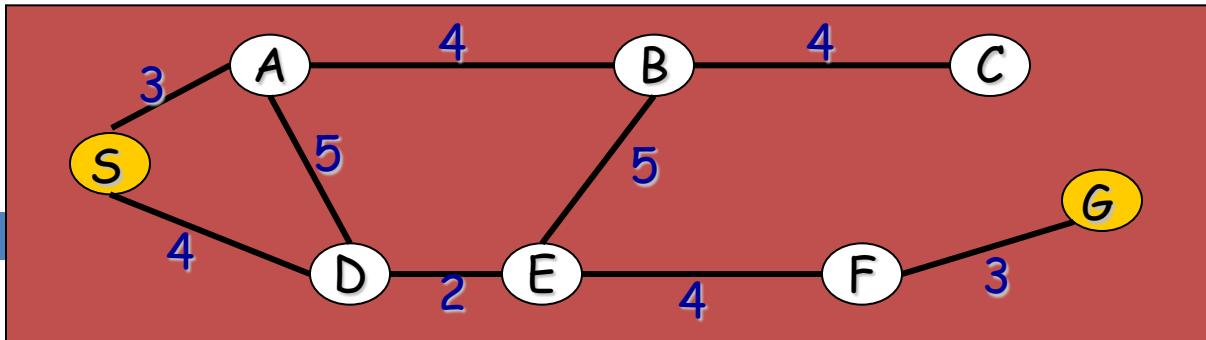
- A depth-first search (DFS) explores a path all the way to a leaf before backtracking and exploring another path
- For example, after searching A, then B, then D, the search backtracks and tries another path from B
- Nodes are explored in the order A B D E H L M N I O P C F G J K Q
- N will be found before J

Depth-First Algorithm:

1. **QUEUE** \leftarrow path only containing the root;
2. **WHILE** { **QUEUE** is not empty
 AND goal is not reached

DO { remove the first path from the **QUEUE**;
 create new paths (to all children);
 reject the new paths with loops;
 add the new paths to front of **QUEUE**;

 
3. **IF** goal reached
 THEN success;
 ELSE failure;



1. **QUEUE** \leftarrow path only containing the root;
2. **WHILE** $\begin{cases} \text{QUEUE is not empty} \\ \text{AND goal is not reached} \end{cases}$
 - DO** $\begin{cases} \text{remove the first path from the QUEUE;} \\ \text{create new paths (to all children);} \\ \text{reject the new paths with loops;} \\ \text{add the new paths to front of QUEUE;} \end{cases}$
3. **IF** goal reached
THEN success;
ELSE failure;

Trace of Depth-First for running example:

- (S) S removed, (SA,SD) computed and added
- (SA, SD) SA removed, (SAB,SAD,SAS) computed,
SAB,SAD) added
- (SAB,SAD,SD) SAB removed, (SABA,SABC,SABE) computed,
(SABC,SABE) added
- (SABC,SABE,SAD,SD) SABC removed, (SABCB) computed,
nothing added
- (SABE,SAD,SD) SABE removed, (SABEB,SABED,SABEF)
computed, (SABED,SABEF)added
- (SABED,SABEF,SAD,SD) SABED removed,
(SABEDS,SABEDA,SABEDE) computed,
nothing added
- (SABEF,SAD,SD) SABEF removed, (SABFE,SABFG)
computed, (SABFG) added
- (SABFG,SAD,SD) goal is reached: reports success

Evaluation Criteria:

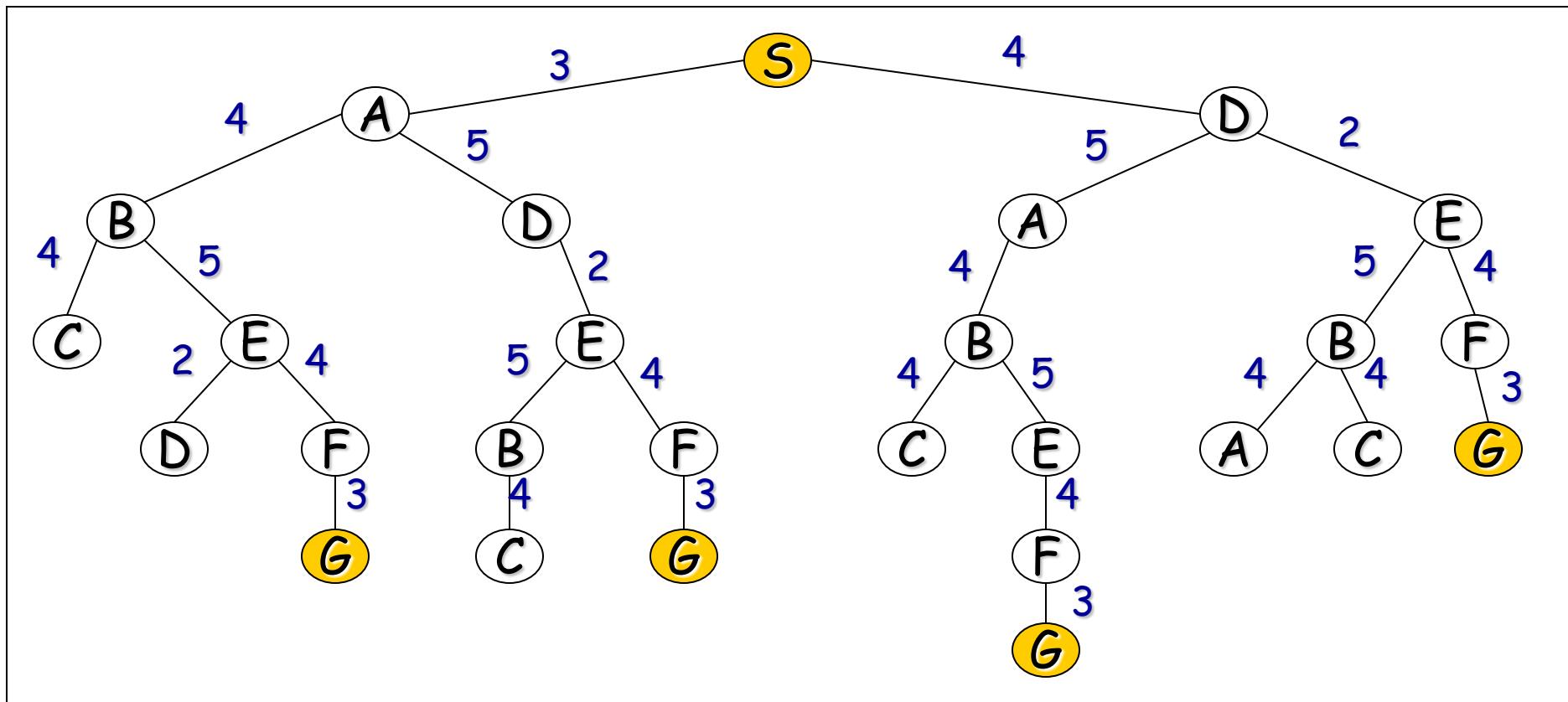
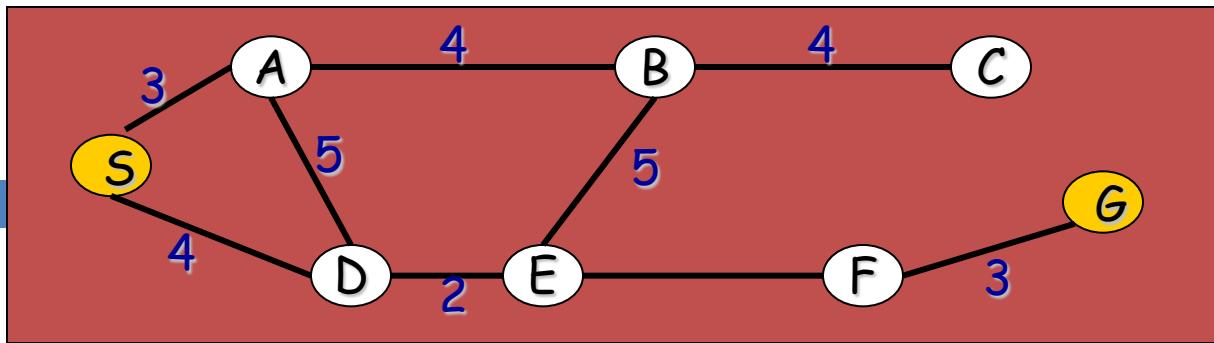
- **Completeness**
 - Does the algorithm always find a path?
 - (for every state space such that a path exists)
- **Speed** (worst time complexity) :
 - What is the highest number of nodes that may need to be created?
- **Memory** (worst space complexity) :
 - What is the largest amount of nodes that may need to be stored?
- Expressed in terms of:
 - d = depth of the tree
 - b = (average) branching factor of the tree
 - m = depth of the shallowest solution

Note: approximations !!

- In our complexity analysis, we do not take the built-in **loop-detection** into account
- The results only ‘formally’ apply to the variants of our algorithms **WITHOUT** loop-checks
- Studying the effect of the loop-checking on the complexity is hard:
 - The overhead of the checking MAY or MAY NOT be compensated by the reduction of the size of the tree
- Also: our analysis **DOES NOT** take the length (space) of representing paths into account !!

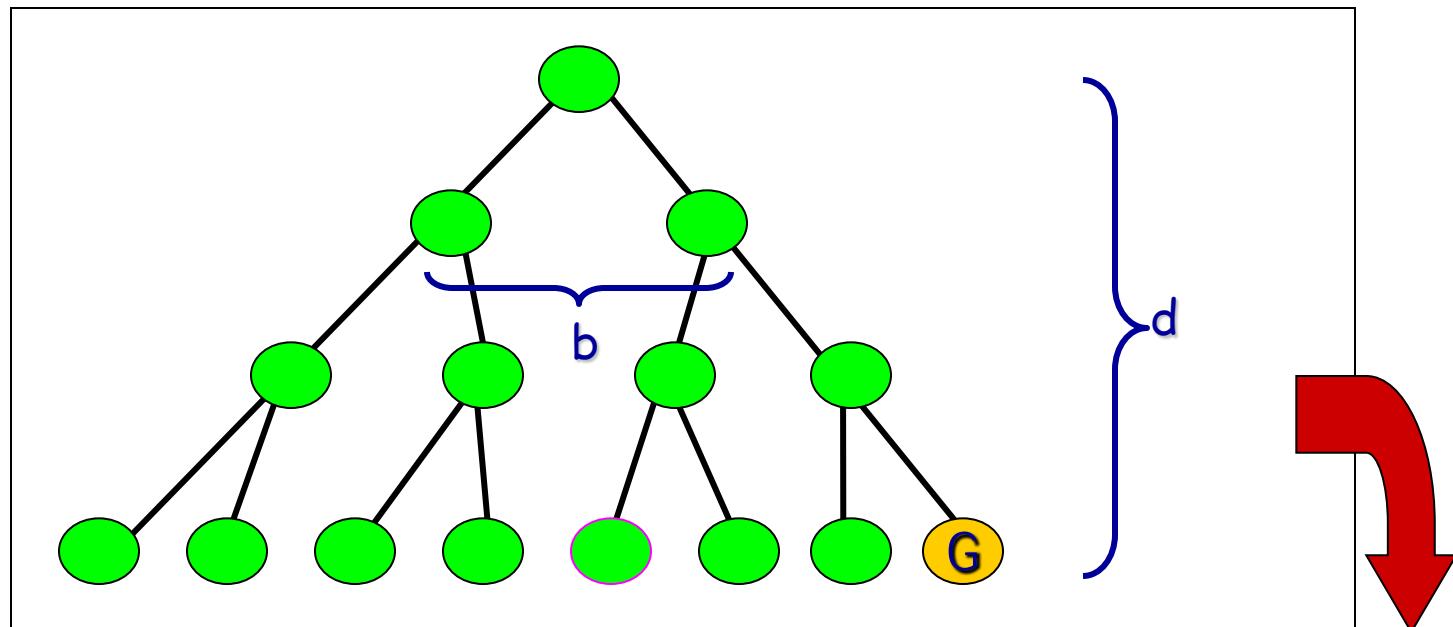
Completeness (depth-first)

- Complete for FINITE (implicit) NETS
 - ▣ (= State space with finitely many nodes)
- **IMPORTANT:**
 - ▣ This is due to integration of LOOP-checking in this version of Depth-First (and in all other algorithms that will follow) !
 - IF we do not remove paths with loops, then Depth-First is not complete (may get trapped in loops of a finite State space)
- **Note:** does NOT find the shortest path



Speed (depth-first)

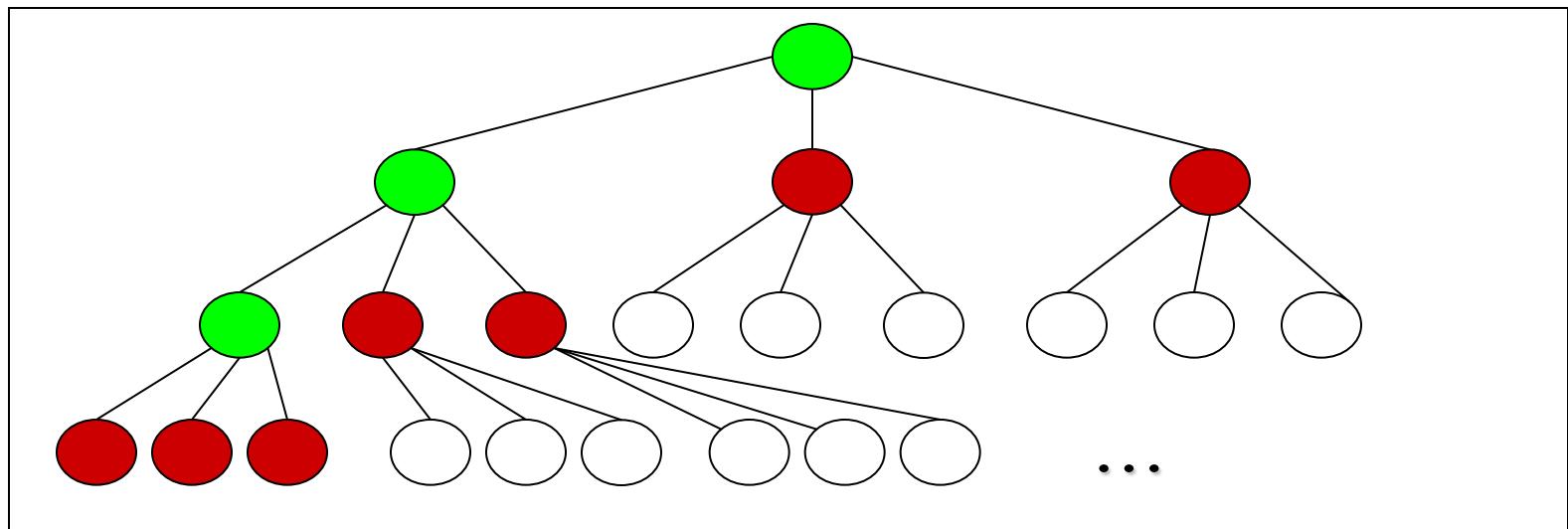
- In the worst case:
 - the (only) goal node may be on the right-most branch,



- Time complexity $= b^d + b^{d-1} + \dots + 1 = \frac{b^{d+1} - 1}{b - 1}$
- Thus: $O(b^d)$

Memory (depth-first)

- Largest number of nodes in QUEUE is reached in bottom left-most node
- Example: $d = 3, b = 3$:

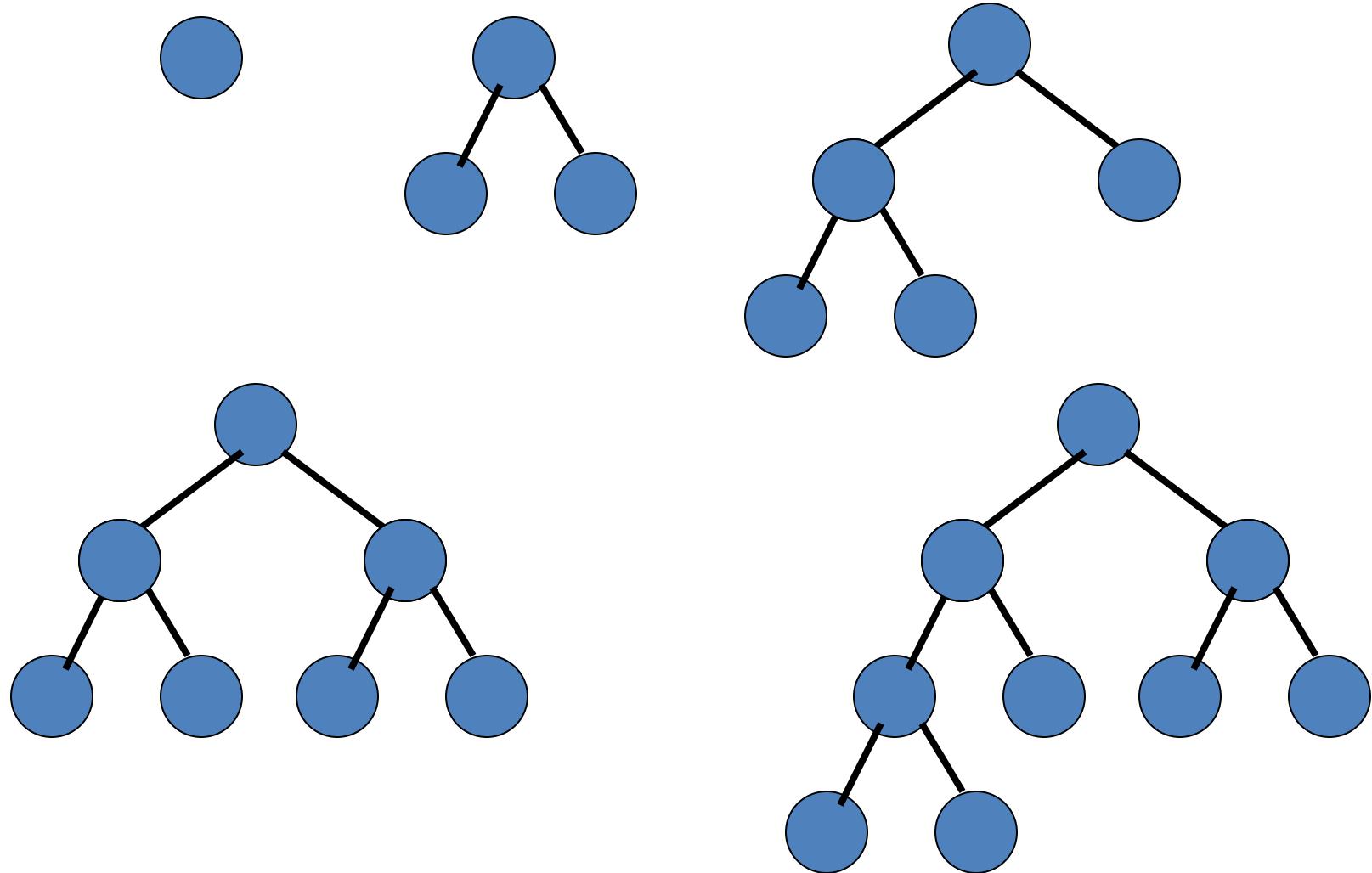


- QUEUE contains all  nodes. Thus: 7.
- In General: $((b-1) * d) + 1$
- Order: $O(d*b)$

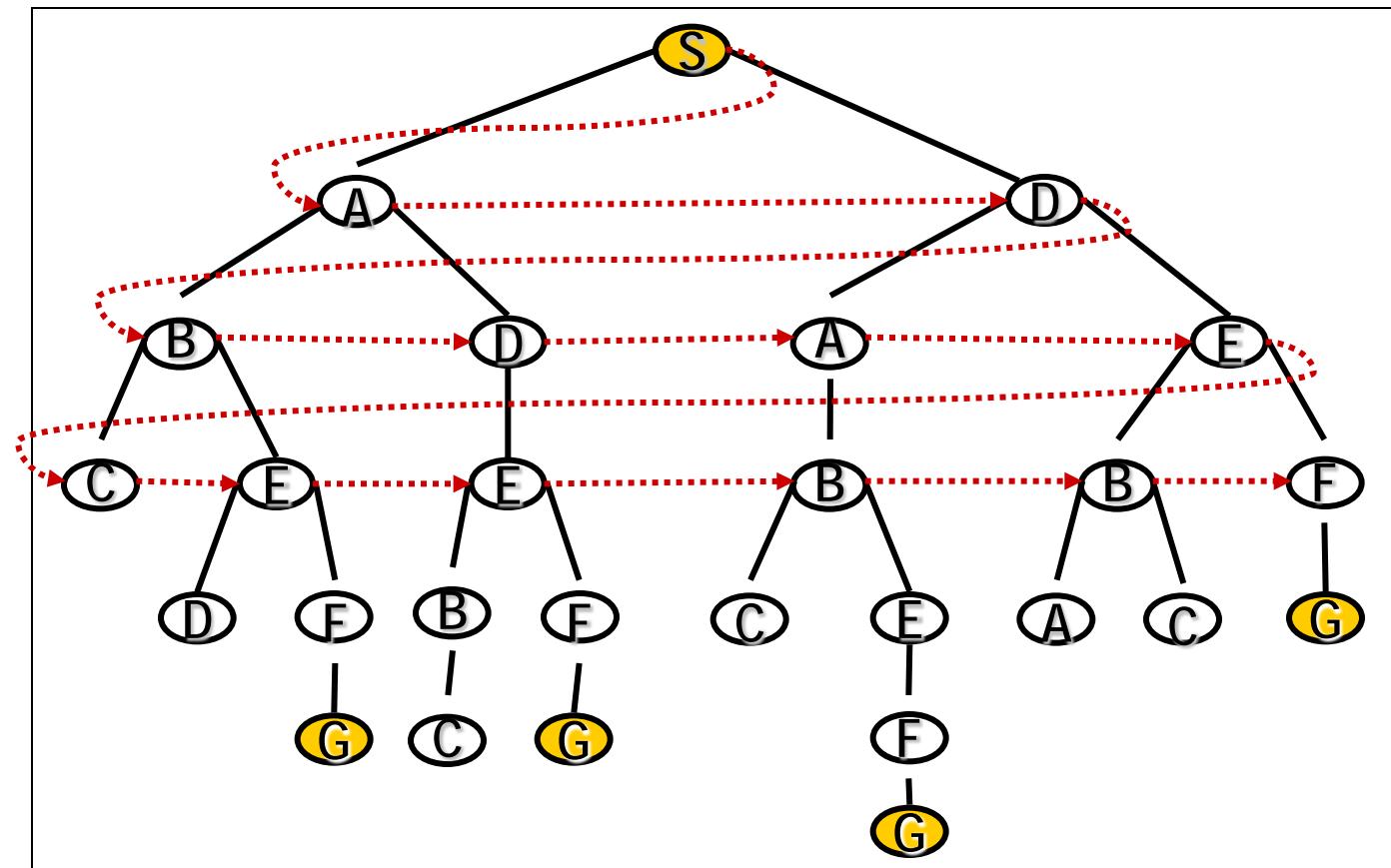
2. Breadth-First Search

- Expand the tree layer by layer, progressing in depth.
- In other words,
 - Expand root node first
 - Expand all nodes at level 1 before expanding level 2
- OR
 - Expand all nodes at level d before expanding nodes at level d+1

Breadth-First Search

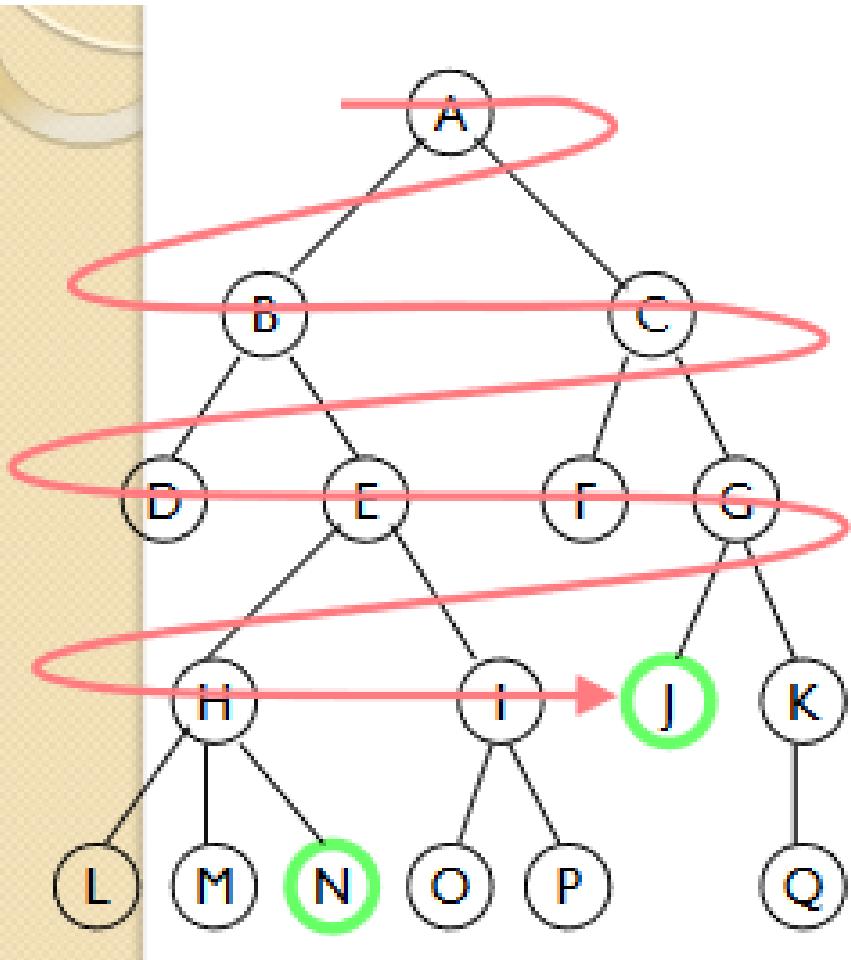


Breadth-First Search:



□ Move downwards, level by level, until goal is reached

Breadth-First Search:



- A breadth-first search (BFS) explores nodes nearest the root before exploring nodes further away
- For example, after searching A, then B, then C, the search proceeds with D, E, F, G
- Nodes are explored in the order A B C D E F G H I J K L M N O P Q
- J will be found before N

Breadth-First Algorithm:

1. **QUEUE** \leftarrow path only containing the root;
 2. WHILE { **QUEUE** is not empty
AND goal is not reached

DO { remove the first path from the **QUEUE**;
create new paths (to all children);
reject the new paths with loops;
add the new paths to back of **QUEUE**;
 3. IF goal reached
THEN success;
ELSE failure;
- 
- 
- ONLY
DIFFERENCE !

Trace of breadth-first for running example:

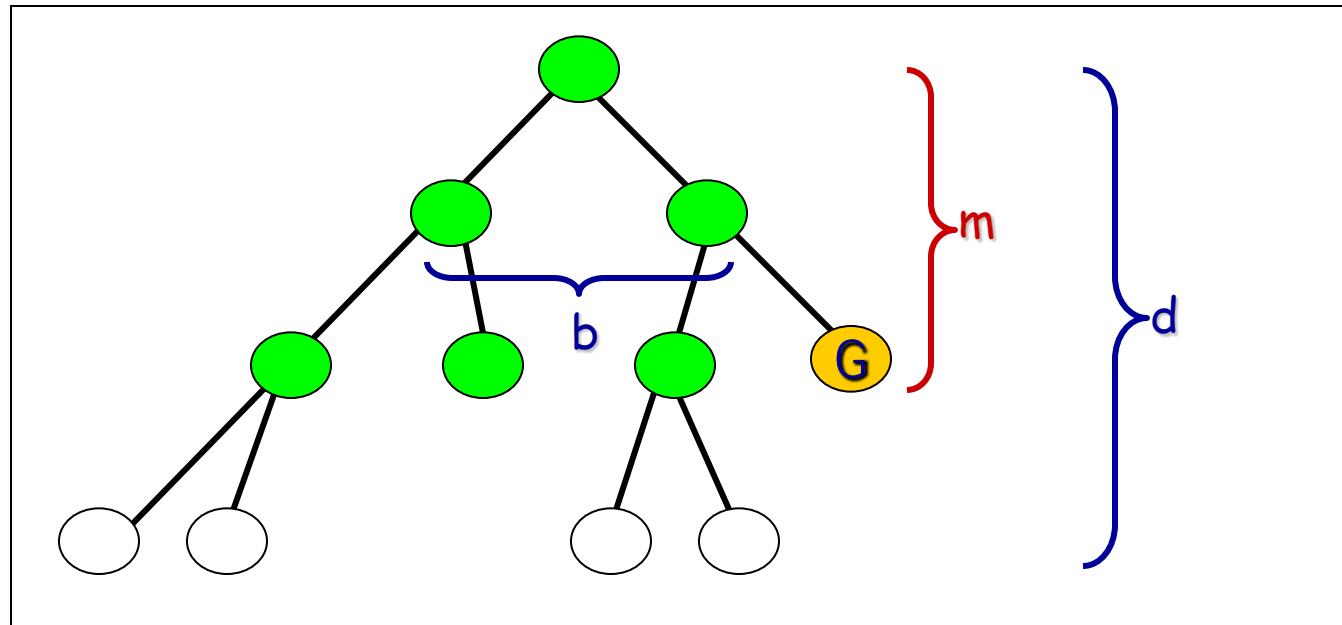
- (S) S removed, (SA,SD) computed and added
- (SA, SD) SA removed, (SAB,SAD,SAS) computed,
(SAB,SAD) added
- (SD,SAB,SAD) SD removed, (SDA,SDE,SDS) computed,
(SDA,SDE) added
- (SAB,SAD,SDA,SDE) SAB removed, (SABA,SABE,SABC)
computed, (SABE,SABC) added
- (SAD,SDA,SDE,SABC,SABE) SAD removed, (SADS,SADA, SADE)
computed, (SADE) added
- etc, until QUEUE contains:
 - (SABED,SABEF,SADEB,SADEF,SDABC,SDABE,SDEBA,SDEBC, SDEFG)
goal is reached: reports success

Completeness (breadth-first)

- **Complete**
 - even for infinite implicit NETS !
 - Would even remain complete without our loop-checking
- **Note:** ALWAYS finds the shortest path

Speed (breadth-first)

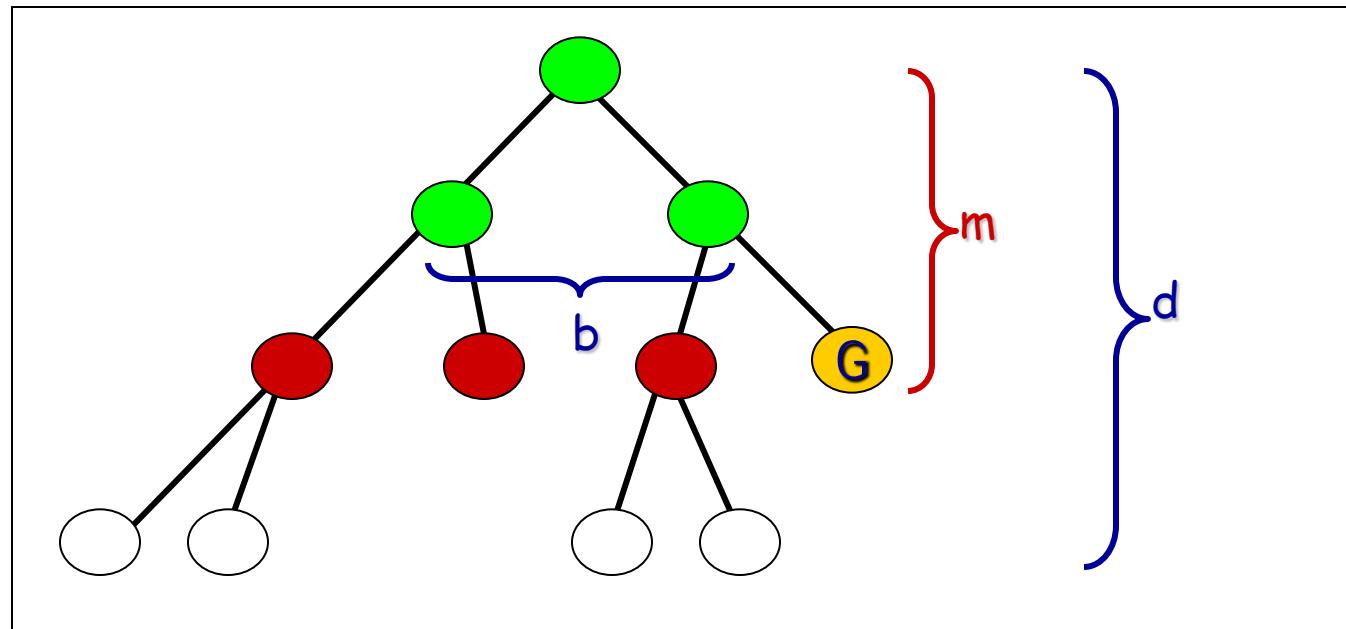
- If a goal node is found on depth m of the tree, all nodes up till that depth are created



- Thus: $O(b^m)$
- **Note:** depth-first would also visit deeper nodes

Memory (breadth-first)

- Largest number of nodes in QUEUE is reached on the level m of the goal node



- QUEUE contains all  and  nodes. (Thus: 4)
- In General: b^m
- This usually is MUCH worse than depth-first !!

Exponential Growth (breadth-first)

Depth	Nodes	Time		Memory	
0	1	1	millisecond	100	kbytes
2	111	0.1	second	11	kilobytes
4	11,111	11	seconds	1	megabyte
6	10^6	18	minutes	111	megabytes
8	10^8	31	hours	11	gigabytes
10	10^{10}	128	days	1	terabyte
12	10^{12}	35	years	111	terabytes
14	10^{14}	3500	years	11,111	terabytes

- Time and memory requirements for breadth-first search, assuming a branching factor of 10, 100 bytes per node and searching 1000 nodes/second

Exponential Growth - Breadth-First Observations

- Space is more of a factor to breadth first search than time
- Time is still an issue. Who has 35 years to wait for an answer to a level 12 problem (or even 128 days to a level 10 problem)
- It could be argued that as technology gets faster then exponential growth will not be a problem. But even if technology is 100 times faster we would still have to wait 35 years for a level 14 problem and what if we hit a level 15 problem!

Practical Evaluation:

- **1. Depth-first search:**
 - IF the search space contains very deep branches without solution, THEN Depth-first may waste much time in them
- **2. Breadth-first search:**
 - Is VERY demanding on memory !
- **Solutions ??**
 - Non-deterministic search
 - Iterative deepening

3. Non-deterministic Search

- A Non-deterministic algorithm is an algorithm that, even for the same input, can exhibit different behaviors on different runs, as opposed to a deterministic algorithm
- There are several ways an algorithm may behave differently from run to run

Non-deterministic Search

1. **QUEUE** \leftarrow path only containing the root;
 2. WHILE { **QUEUE** is not empty
 AND goal is not reached

DO { remove the first path from the **QUEUE**;
 create new paths (to all children);
 reject the new paths with loops;
 add the new paths in random places in **QUEUE**;
 3. IF goal reached
 THEN success;
 ELSE failure;
- 
- 

4. Iterative Deepening Search

- Also referred to as **Iterative Deepening Depth-First Search**
- Restrict a depth-first search to a fixed depth
 - a depth-limited version of depth-first search is run repeatedly with increasing depth limits until the goal is found
- If no path is found, increase the depth and restart the search

Depth-limited Search

1. DEPTH <-- <some natural number>
QUEUE <-- path only containing the root;
2. WHILE { QUEUE is not empty
AND goal is not reached
DO { remove the first path from the QUEUE;
~~IF~~ path has length smaller than DEPTH
 create new paths (to all children);
 reject the new paths with loops;
 add the new paths to front of QUEUE;
3. IF goal reached
 THEN success;
 ELSE failure;

Iterative Deepening Algorithm:

1. DEPTH \leftarrow 1
2. WHILE goal is not reached
 - DO { perform Depth-limited search;
 - { increase DEPTH by 1;

Iterative Deepening: the best 'blind' search

- Complete: yes - even finds the shortest path (like breadth first)
- Memory: b^m (combines advantages of depth- and breadth-first)
- Speed:
 - If the path is found for Depth = m , then how much time was wasted constructing the smaller trees??
- $b^{m-1} + b^{m-2} + \dots + 1 = \frac{b^m - 1}{b - 1} = O(b^{m-1})$
- While the work spent at DEPTH = m itself is $O(b^m)$



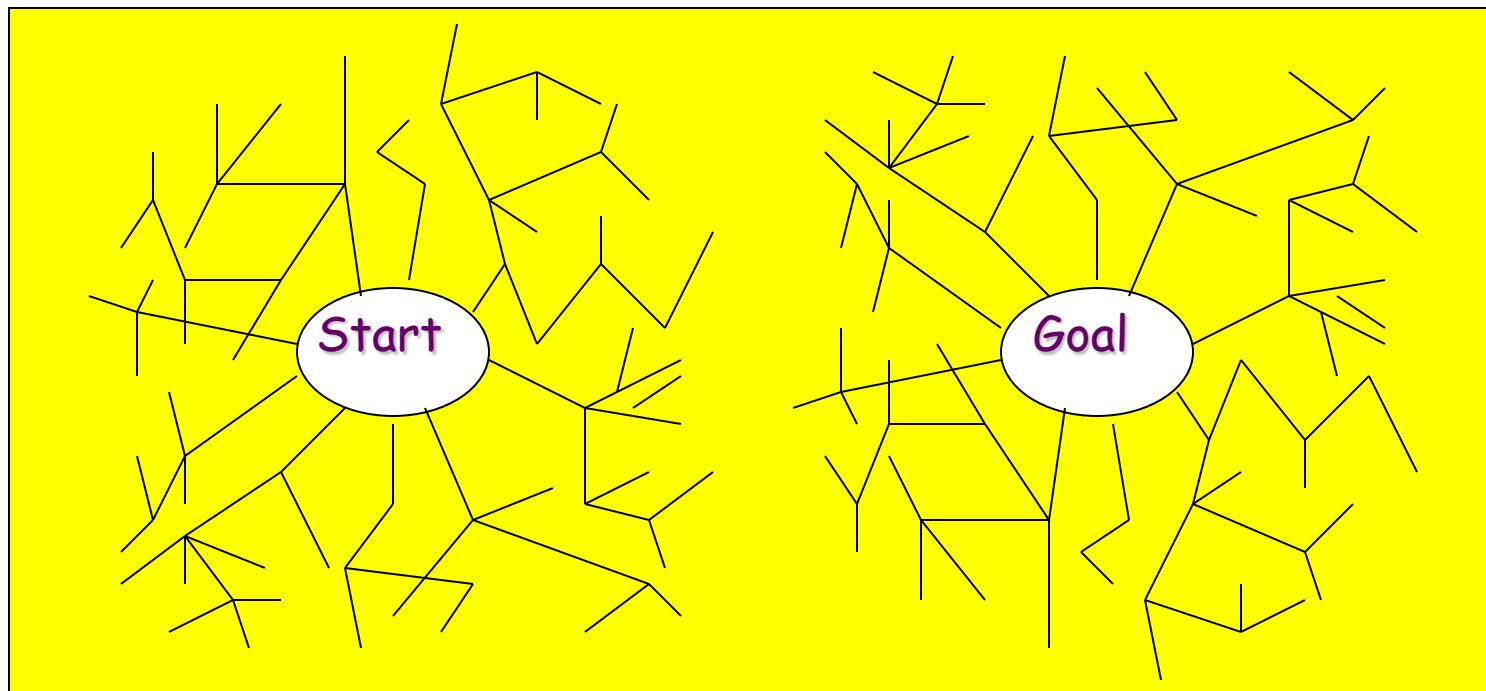
In general: **VERY good trade-off**

5. Bi-directional Search

- Compute the tree from the start node and from a goal node, until these meet

Bi-directional Search

- **IF** you are able to EXPLICITLY describe the GOAL state, **AND** you have BOTH rules for FORWARD reasoning AND BACKWARD reasoning:



Bi-directional Algorithm:

1. $\text{QUEUE1} \leftarrow$ path only containing the root;
 $\text{QUEUE2} \leftarrow$ path only containing the goal;
2. WHILE both QUEUE_i are not empty
AND QUEUE1 and QUEUE2 do NOT share a state

DO {
remove their first paths;
create their new paths (to all children);
reject their new paths with loops;
add their new paths to back;
}
3. IF QUEUE1 and QUEUE2 share a state
THEN success;
ELSE failure;

Properties (Bi-directional):

- Complete: Yes.
- Speed: If the test on common state can be done in constant time (hashing):
 - $2 * O(b^{m/2}) = O(b^{m/2})$
- Memory: similarly: $O(b^{m/2})$

Exercise

- Confirm the trace of depth-first search algorithm given in slide 11
- Confirm the trace of breadth-first search algorithm given in slide 22
- Uniform Cost Search: find out how this search algorithm works

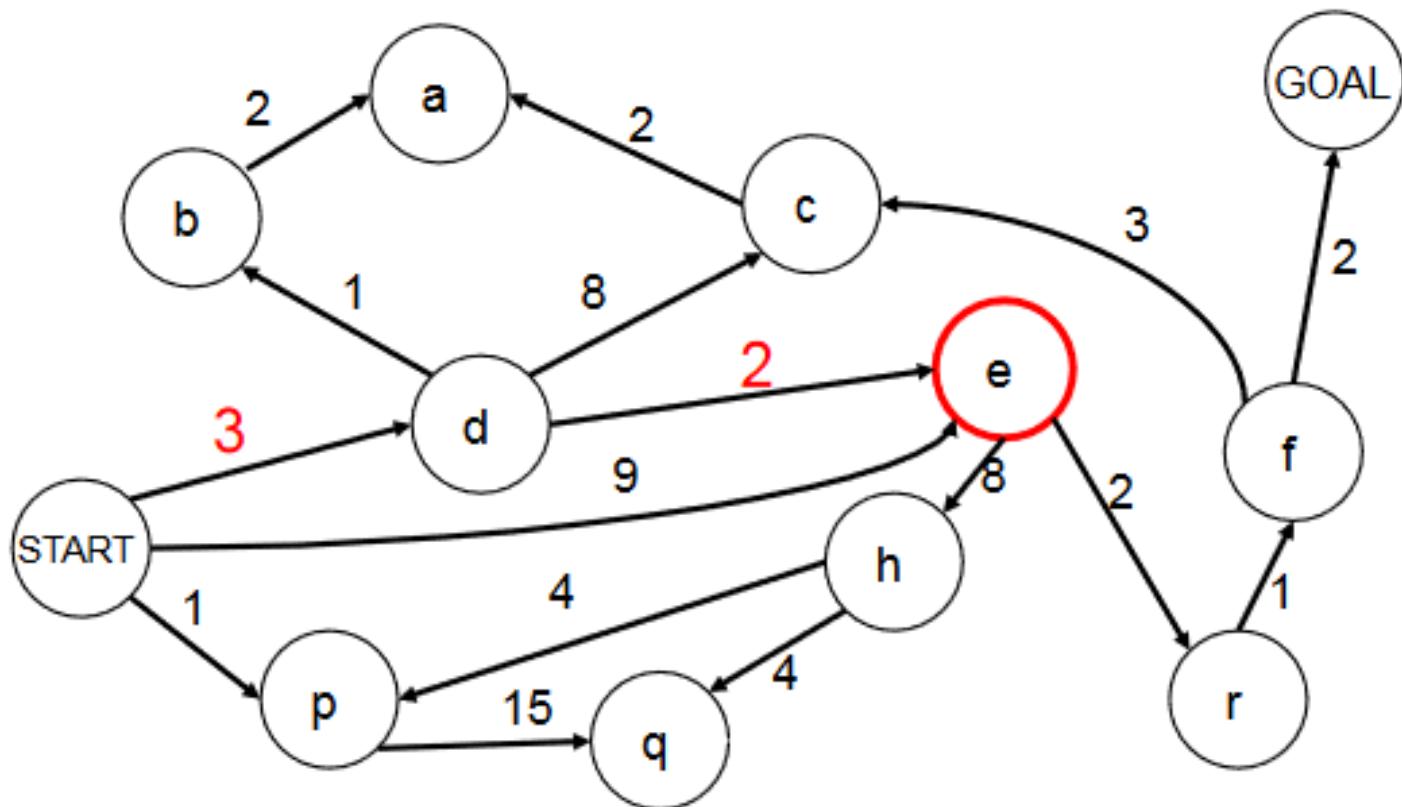
Uniform-Cost Search Algorithm

- If all the edges in the search graph do not have the same cost then breadth-first search generalizes to uniform-cost search. Instead of expanding nodes in order of their depth from the root, uniform-cost search expands nodes in order of their cost from the root. At each step, the next step n to be expanded is one whose cost $g(n)$ is lowest where $g(n)$ is the sum of the edge costs from the root to node n . The nodes are stored in a priority queue. This algorithm is also known as Dijkstra's single-source shortest algorithm.
- UCS is Dijkstra's algorithm which is focused on finding a single shortest path to a single finishing point rather than the shortest path to every point. UCS does this by stopping as soon as the finishing point is found.
- Whenever a node is chosen for expansion by uniform cost search, a lowest-cost path to that node has been found. The worst case time complexity of uniform-cost search is $O(b^c/m)$, where c is the cost of an optimal solution and m is the minimum edge cost. Unfortunately, it also suggests the same memory limitation as breadth-first search.

Uniform-Cost Search Algorithm

- with $f(n)$ = the sum of edge costs from start to n

$f(n)$ = “cost from **start** to **n**”



COMP 308

ARTIFICIAL INTELLIGENCE

PART 4 – GAME PLAYING

Njeri Ireri

Jan – April 2020

We Shall Discuss

- Games. Why?
- Minimax search
- Alpha-beta pruning

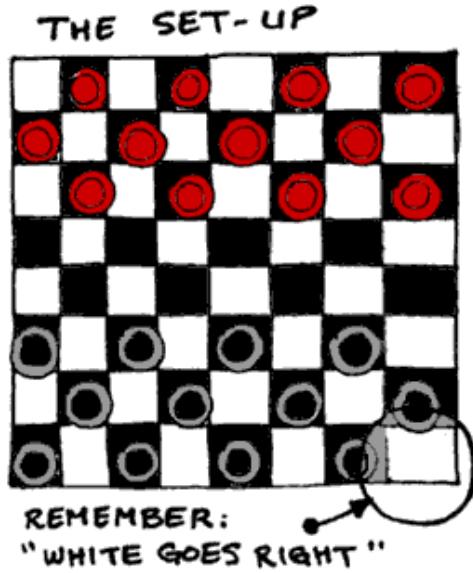
Why are people interested in games?

Some games:

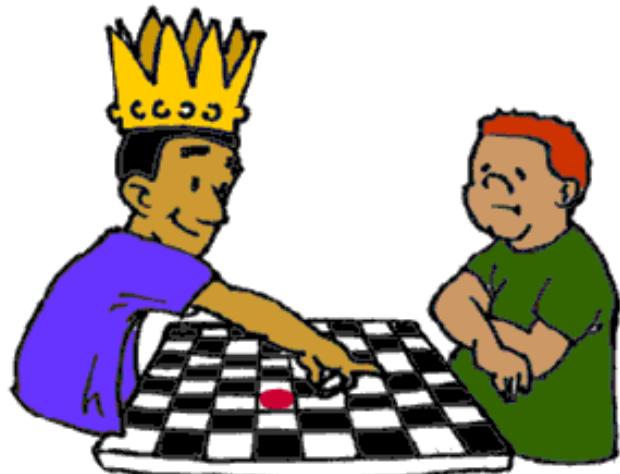
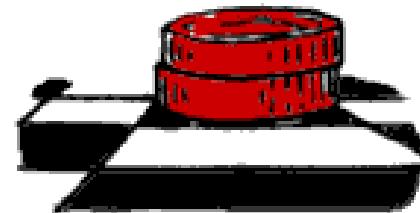
- Ball games
- Card games
- Board games
- Computer games
- ...

Why are people interested in games?

Draughts



"CROWNED"



Is a computer more intelligent
if it beats you in a game of draughts?

Why are people interested in games? Robocup



Why study board games ?

- One of the oldest sub-fields of AI.
- Game Playing has been studied for a long time
 - Babbage (tic-tac-toe)
 - Turing (chess)
- It is an abstract and pure form of competition that seems to require intelligence
- Easy to represent the states and actions
 - Very little world knowledge required !
- Game playing research has contributed ideas on how to make the best use of time to reach good decisions, when reaching optimal decisions is impossible. These ideas are applicable in tackling real-world search problems
 - “A chess (or draughts) playing computer would be proof of a machine doing something thought to require intelligence”

Game playing

- Game playing is a special case of a search problem, with some new requirements
- Up till now we have assumed the situation is not going to change whilst we search
 - static environment but
- Game playing is not like this, because
 - The opponent introduces uncertainty
 - The opponent also wants to win

Our limit here: 2-person games, with no chance

Game playing

- “Contingency” problem:
 - We don’t know the opponents move !
- The size of the search space:
 - Chess : ~15 moves possible per state, 80 ply
 - 15^{80} nodes in tree
 - Go : ~200 moves per state, 300 ply
 - 200^{300} nodes in tree
- Game playing algorithms: they involve
 - Search tree only up to some depth bound
 - Use an evaluation function at the depth bound
 - Propagate the evaluation upwards in the tree

Types of games

	<u>Deterministic</u>	<u>Chance</u>
<u>Perfect information</u>	Chess, draughts, Go, Othello, Tic-tac-toe, Ajua	Backgammon, Monopoly
<u>Imperfect information</u>		Bridge, Poker, Scrabble

Game Playing - Chess

- Shannon - March 9th 1949 - New York
- Size of search space (10^{120} - average of 40 moves)
- 200 million positions/second = 10^{100} years to evaluate all possible games
- Searching to depth = 40, at one node per microsecond it would take 10^{90} years to make its first move

Game Playing - Chess

- 1957 - Newell and Simon predicted that a computer would be chess champion within ten years
- Simon : “I was a little far-sighted with chess, but there was no way to do it with machines that were as slow as the ones way back then”
- 1958 - First computer to play chess was an IBM 704 - about one millionth capacity of deep blue.
- 1967 : Mac Hack competed successfully in human tournaments
- 1983 : “Belle” obtained expert status from the United States Chess Federation
- Mid 80’s : Scientists at Carnegie Mellon University started work on what was to become Deep Blue.
- Project moved to IBM in 1989

Game Playing - Chess

- May 11th 1997, Gary Kasparov lost a six match game to Deep blue
 - 3.5 to 2.5
 - Two wins for deep blue, one win for Kasparov and three draws

Game Playing - Checkers (Draughts)

- Arthur Samuel - 1952
- Written for an IBM 701
- 1954 - Re-wrote for an IBM 704
 - 10,000 words of main memory
- Added a learning mechanism that learnt its own evaluation function
- Learnt the evaluation function by playing against itself
- After a few days it could beat its creator
- And compete on equal terms with strong human players
- Jonathon Schaeffer - 1996
- Developed Chinook

Game Playing - Checkers (Draughts)

- Chinook
- Uses Alpha-Beta search
- Plays a perfect end game by means of a database
- In 1992 Chinook won the US Open
- And challenged for the world championship
- Dr Marion Tinsley, had been world championship for over 40 years
 - ... only losing three games in all that time
 - Against Chinook she suffered her fourth and fifth defeat
 - But ultimately won 21.5 to 18.5

Game Playing - Checkers (Draughts)

- In August 1994 there was a re-match but Marion Tinsley withdrew for health reasons
- Chinook became the official world champion
- Schaeffer claimed Chinook was rated at 2814
- The best human players are rated at 2632 and 2625
- Chinook did not include any learning mechanism

Game Playing - Checkers (Draughts)

- Kumar - 2000
- “Learnt” how to play a good game of checkers
- The program used a *population* of games with the best competing for *survival*
- Learning was done using a *neural network* with the synapses being changed by an *evolutionary strategy*
- The best program beat a commercial application 6-0

- The program was presented at CEC 2000 (San Diego) and remain undefeated

Game Playing - Minimax

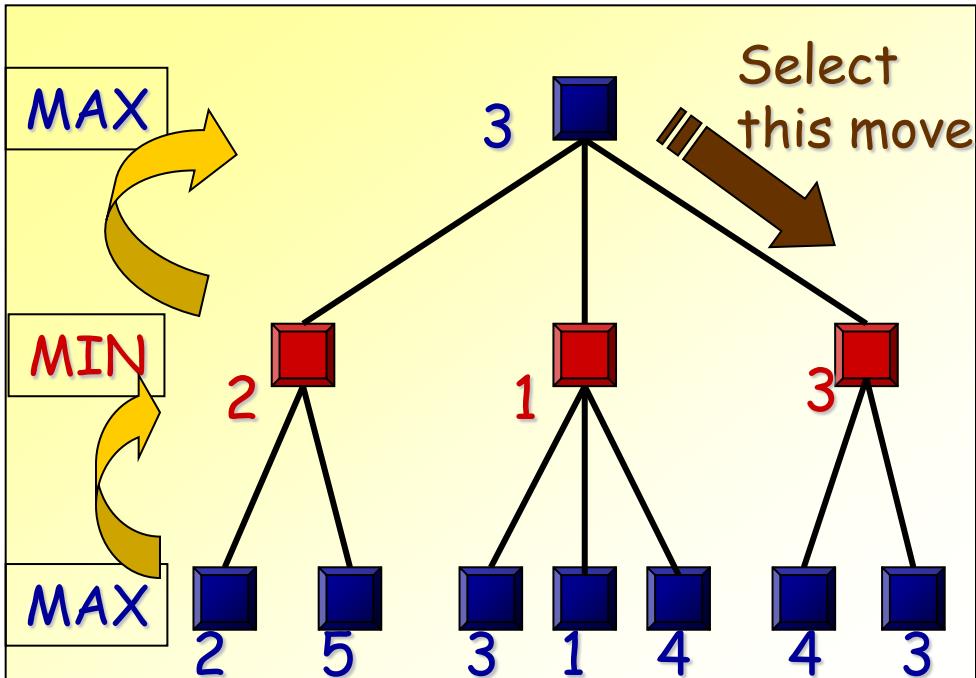
- Game Playing: An opponent tries to thwart your every move
- 1944 - John von Neumann outlined a search method (**Minimax**) that maximised your position whilst minimising your opponents
- In order to implement we need a method of measuring how good a position is
 - Often called a **utility function** (or **payoff function**)
 - e.g. outcome of a game; win 1, loss -1, draw 0
- Initially this will be a value that describes our position exactly

Game Playing - Minimax

- The MiniMax algorithm **selects the “best” next move for a computer player in a two-player game.** The algorithm makes a tree of all possible moves for both players
- This algorithm is called MiniMax simply because the computer makes moves that bring it **maximum gain**, while assuming the opponent makes moves that brings the computer **minimum gain**. Because the players alternate moves, the algorithm alternates between minimizing and maximizing levels of the recursive search tree
- Let's look at a hypothetical search tree to see how the MiniMax analysis works to ultimately select the best move; this example shows a game where there are either two or three possible moves, and where the look ahead limit (search depth) is two moves...

Game Playing - Minimax

- Restrictions:
 - 2 players: MAX (computer) and MIN (opponent)
 - deterministic, perfect information
- Select a depth-bound (say: 2) and evaluation function



- Construct the tree up till the depth-bound
- Compute the evaluation function for the leaves
- Propagate the evaluation function upwards:
 - taking minima in MIN
 - taking maxima in MAX

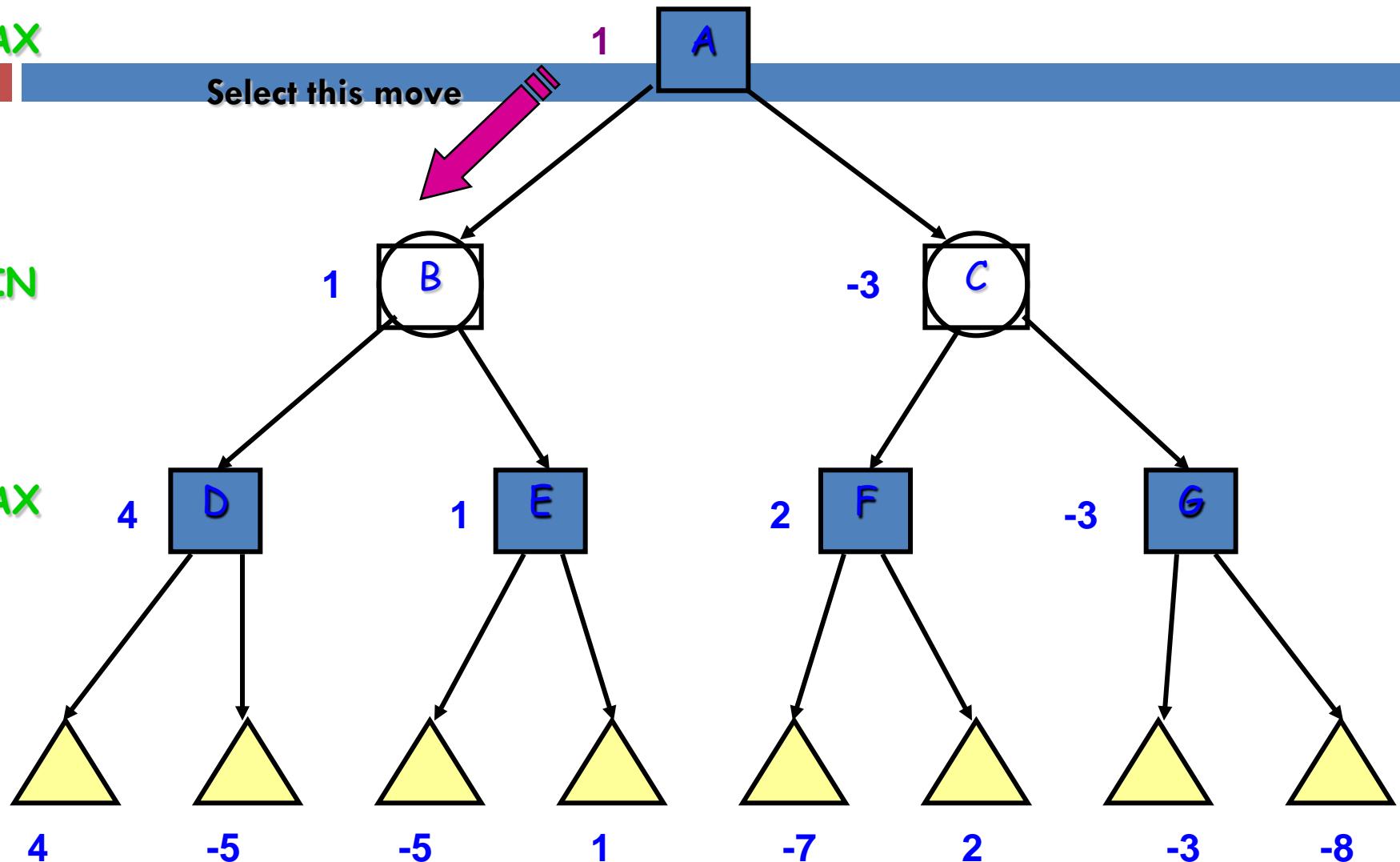
Game Playing - Minimax Example

MAX

Select this move

MIN

MAX



= terminal position



= agent



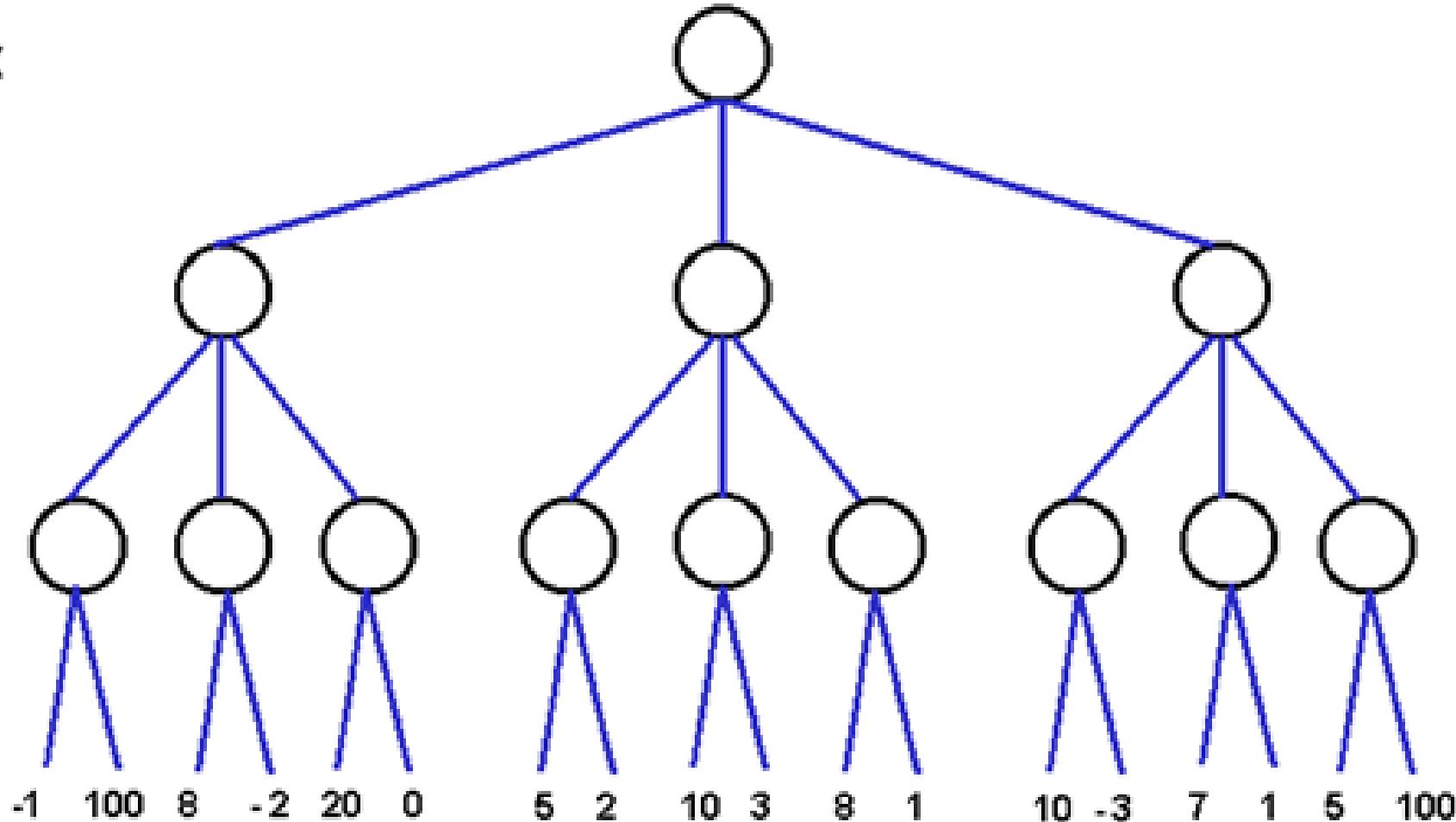
= opponent

Exercise

MAX

MIN

MAX



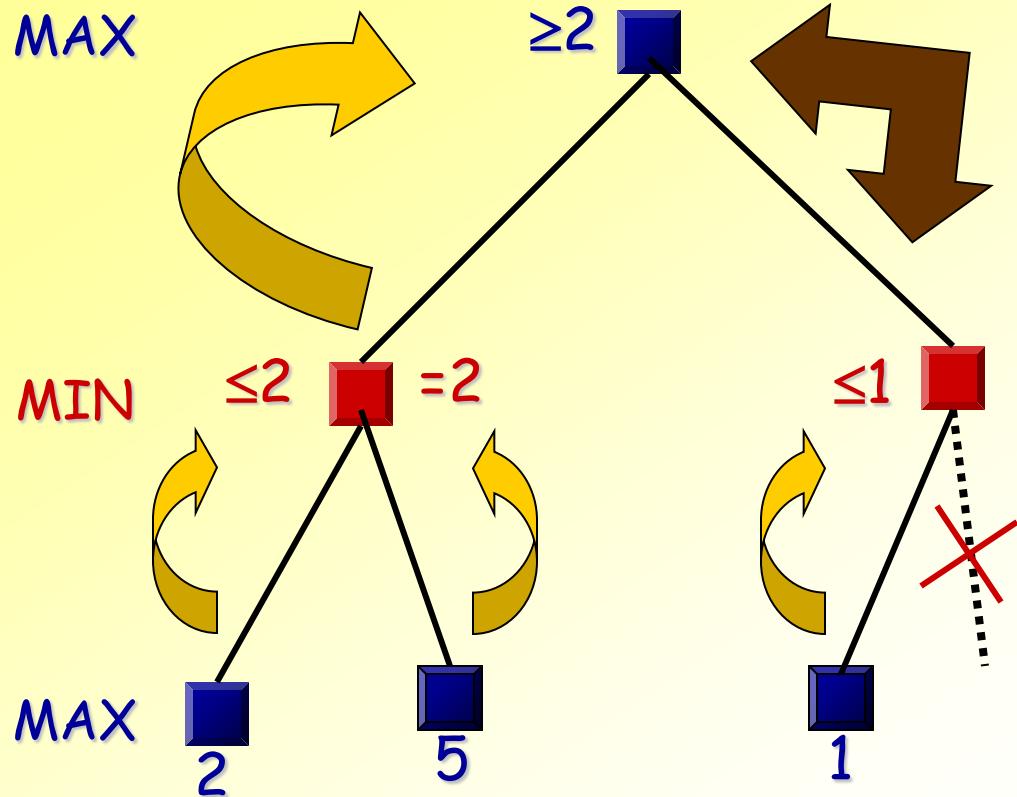
Game Playing - Alpha-Beta Pruning

- Generally applied optimization on Mini-max
- Instead of:
 - first creating the entire tree (up to depth-level)
 - then doing all propagation
- Interleave the generation of the tree and the propagation of values
- Point:
 - some of the obtained values in the tree will provide information that other (non-generated) parts are redundant and do not need to be generated

Alpha-Beta idea:

□ Principles:

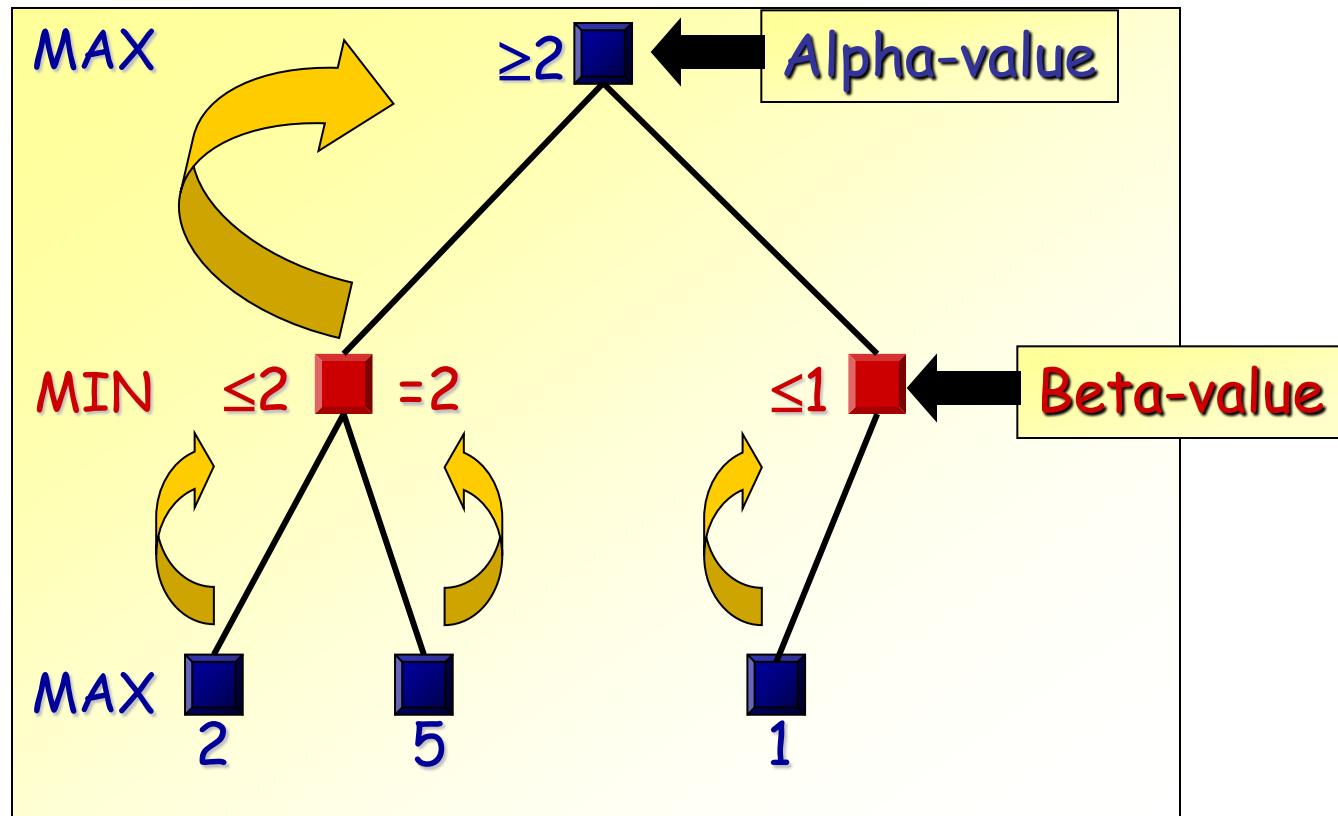
- generate the tree depth-first, left-to-right
- propagate final values of nodes as initial estimates for their parent node



- The **MIN**-value (1) is already smaller than the **MAX**-value of the parent (2)
- The **MIN**-value can only decrease further,
- The **MAX**-value is only allowed to increase,
- No point in computing further below this node

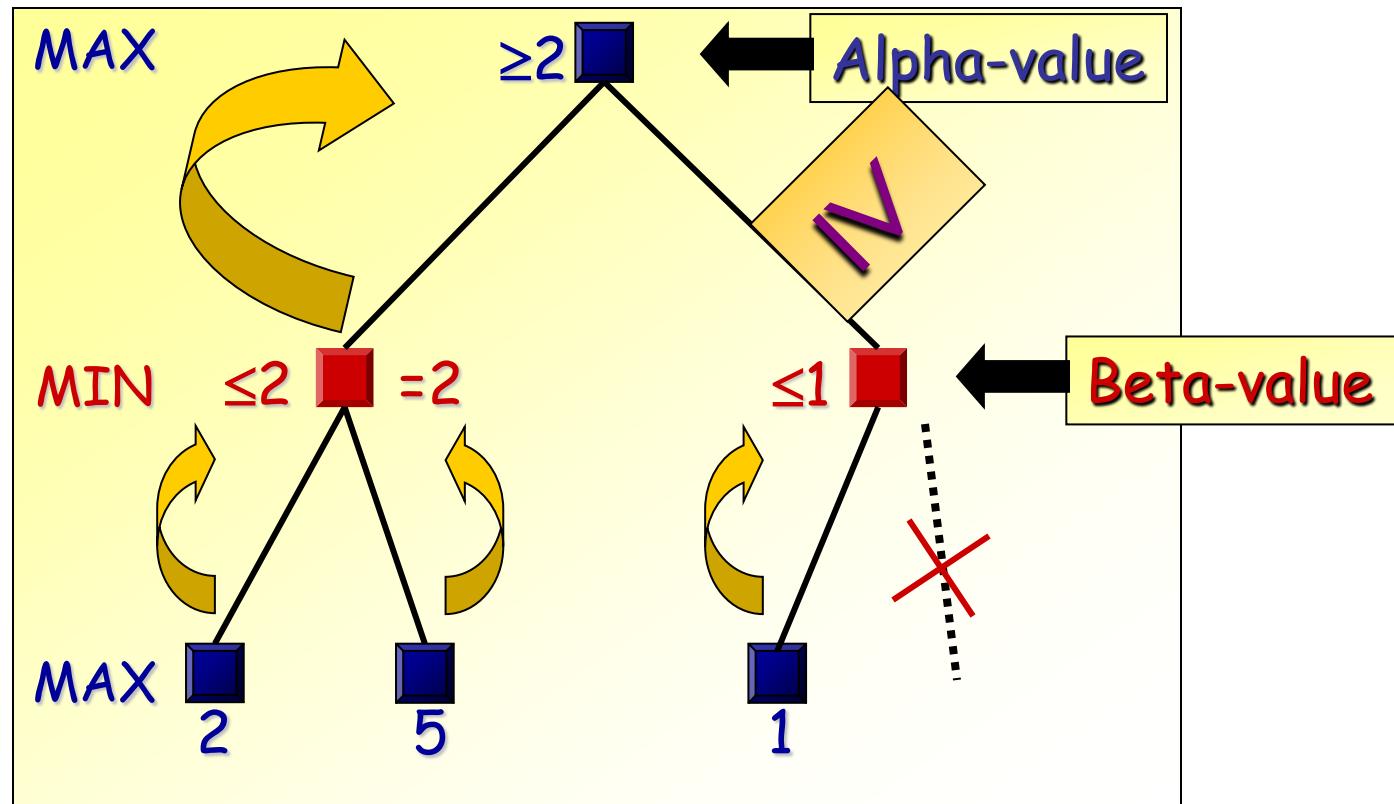
Terminology:

- The (temporary) values at MAX-nodes are ALPHA-values
- The (temporary) values at MIN-nodes are BETA-values



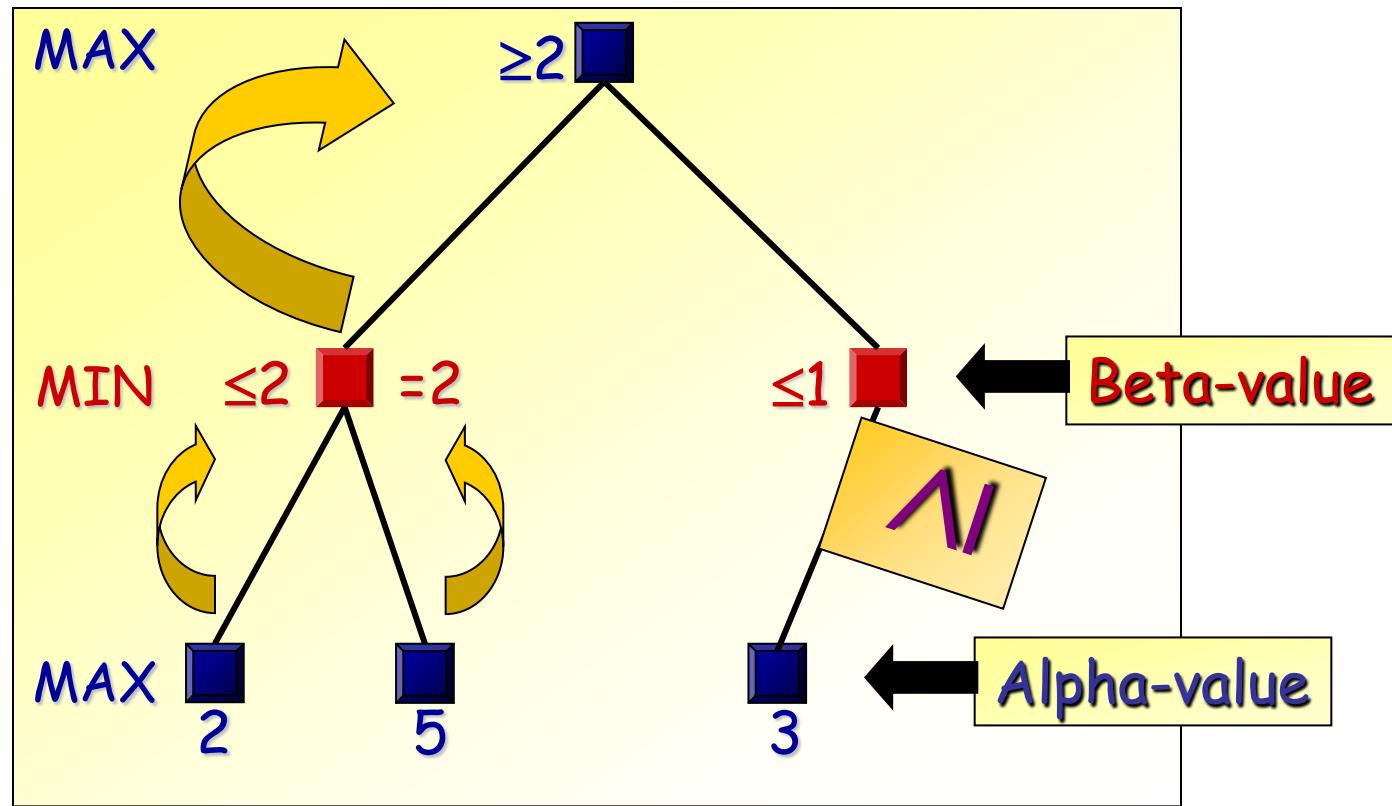
The Alpha-Beta principles (1):

- If an ALPHA-value is larger or equal than the Beta-value of a descendant node:
stop generation of the children of the descendant

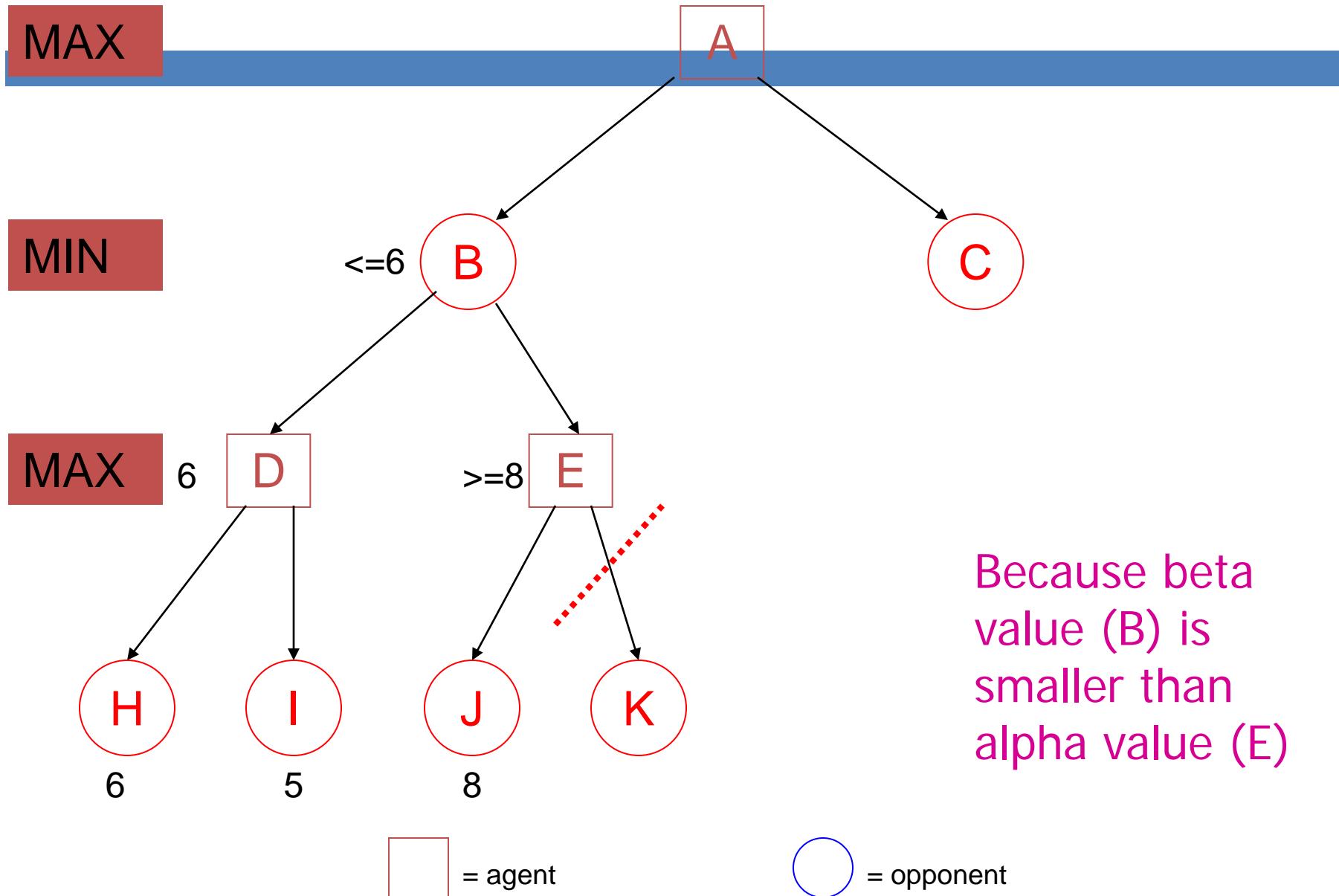


The Alpha-Beta principles (2):

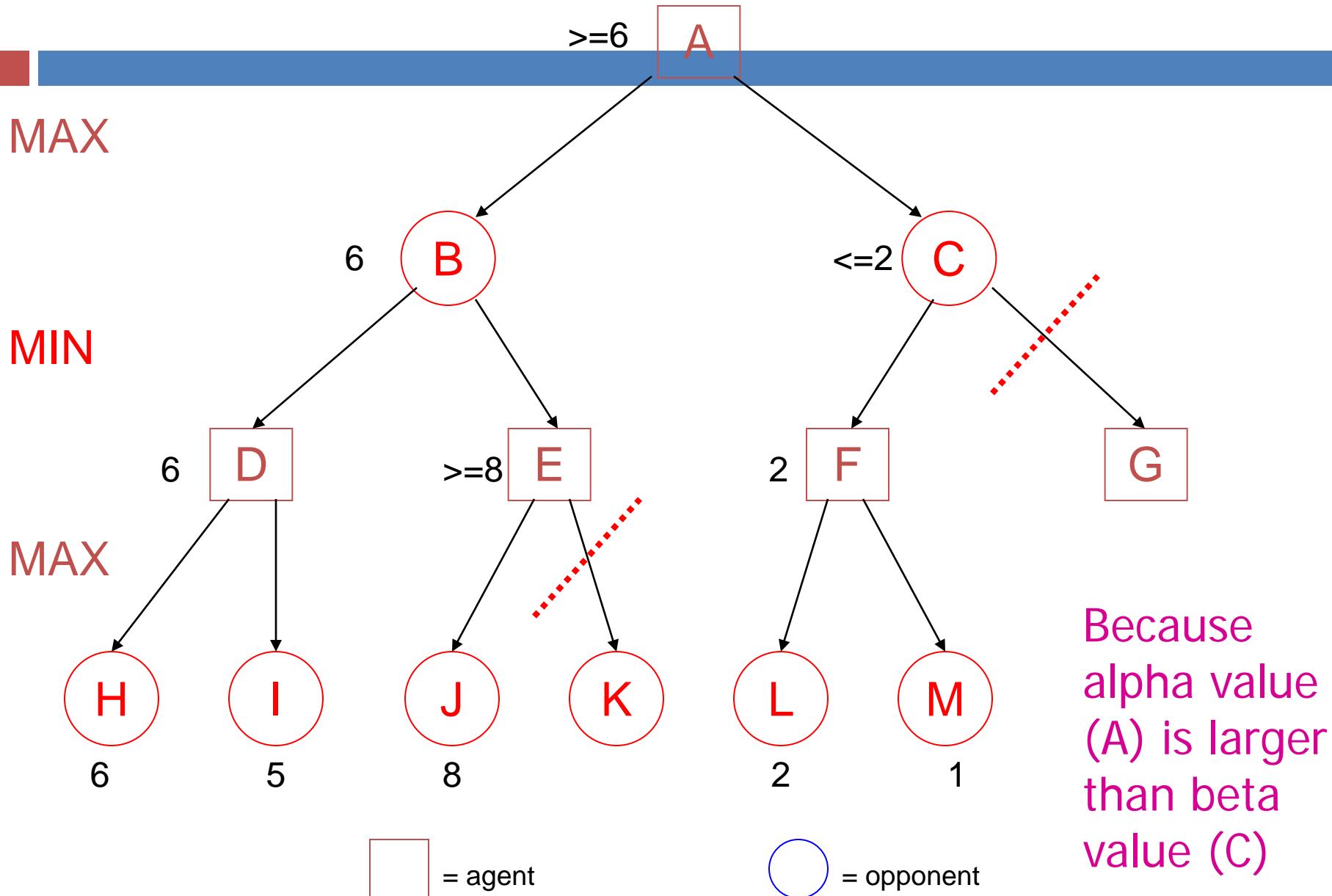
- If an **Beta-value** is **smaller or equal** than the **Alpha-value** of a descendant node:
stop generation of the children of the descendant



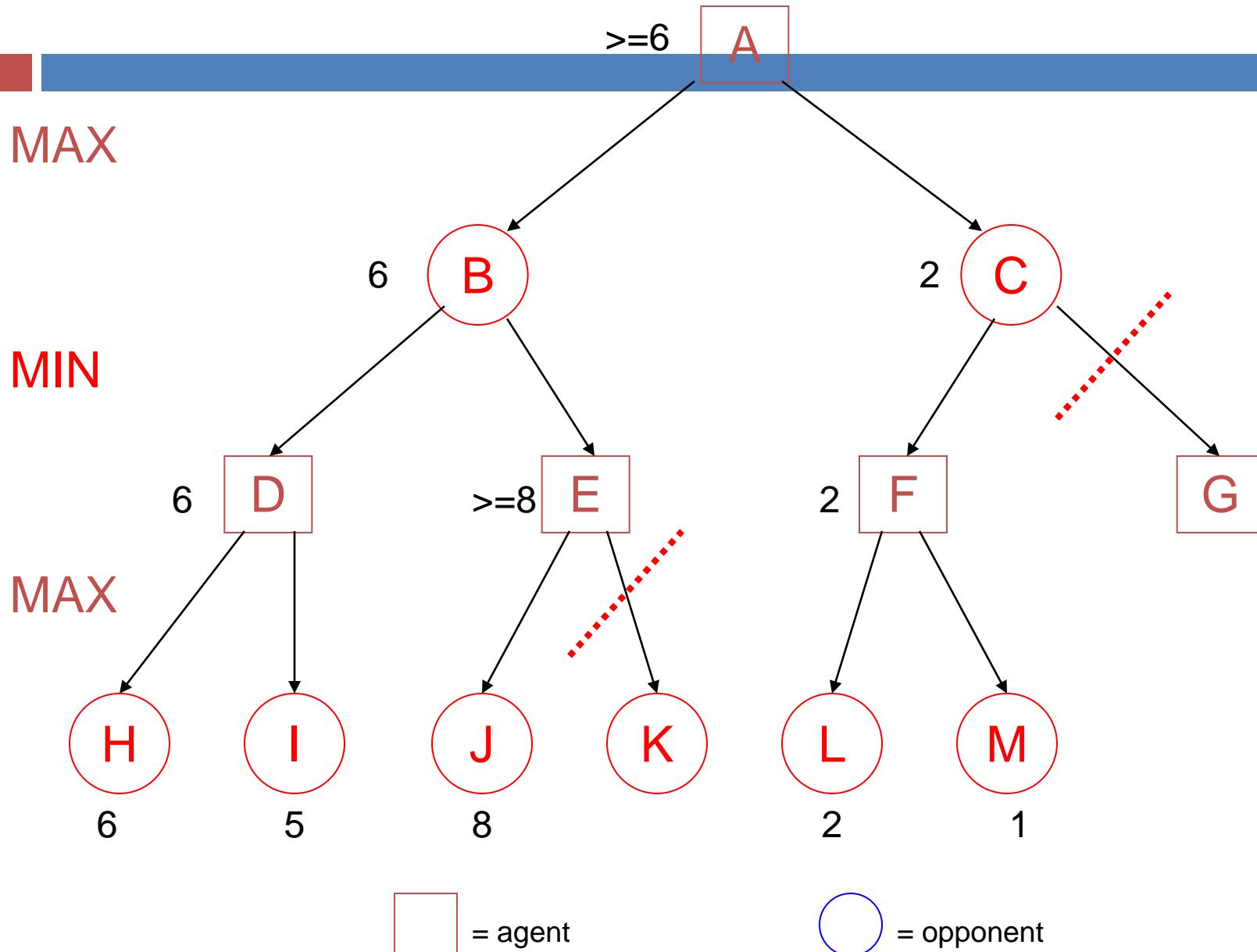
Alpha-Beta Pruning Example



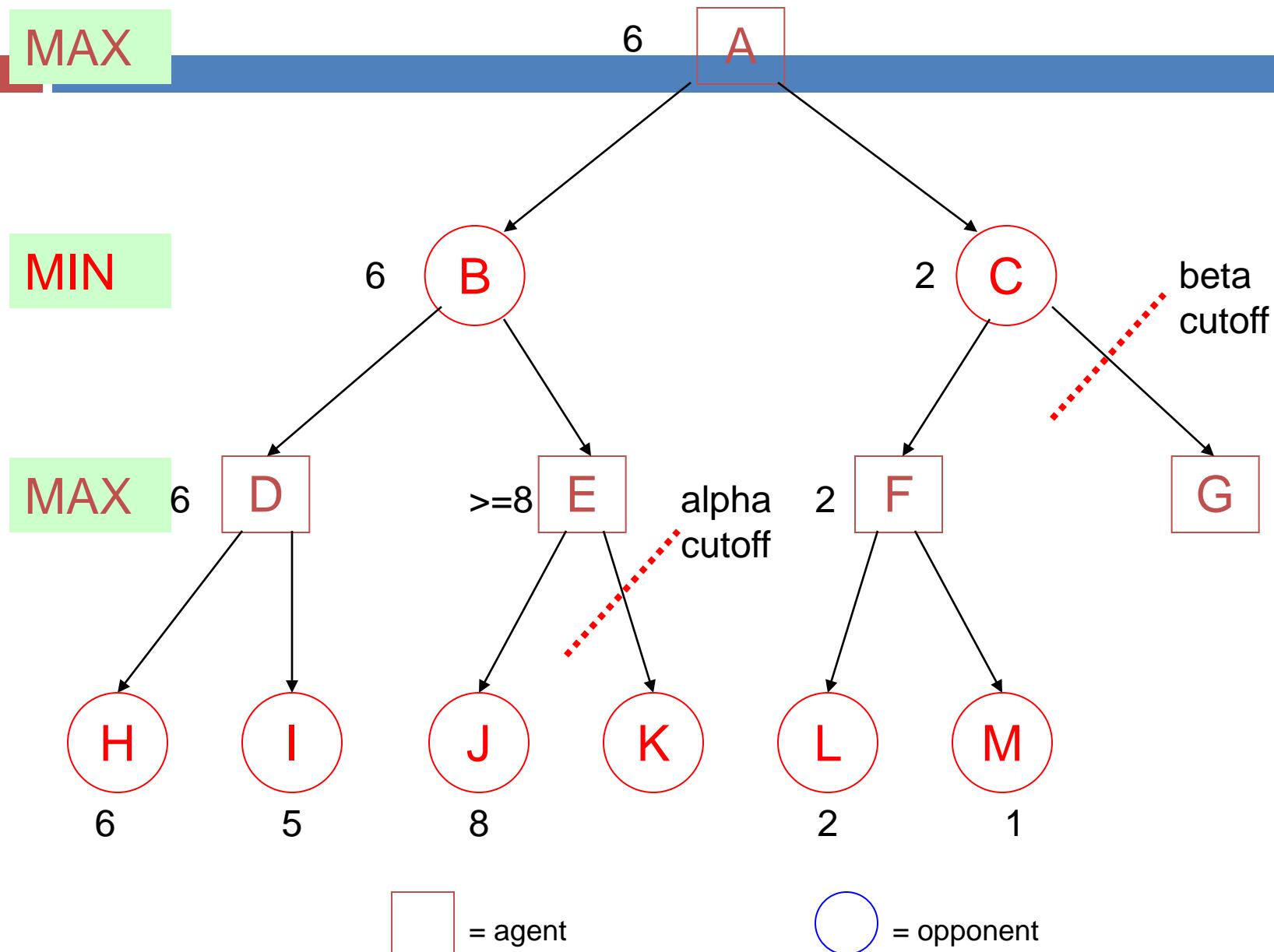
Alpha-Beta Pruning Example



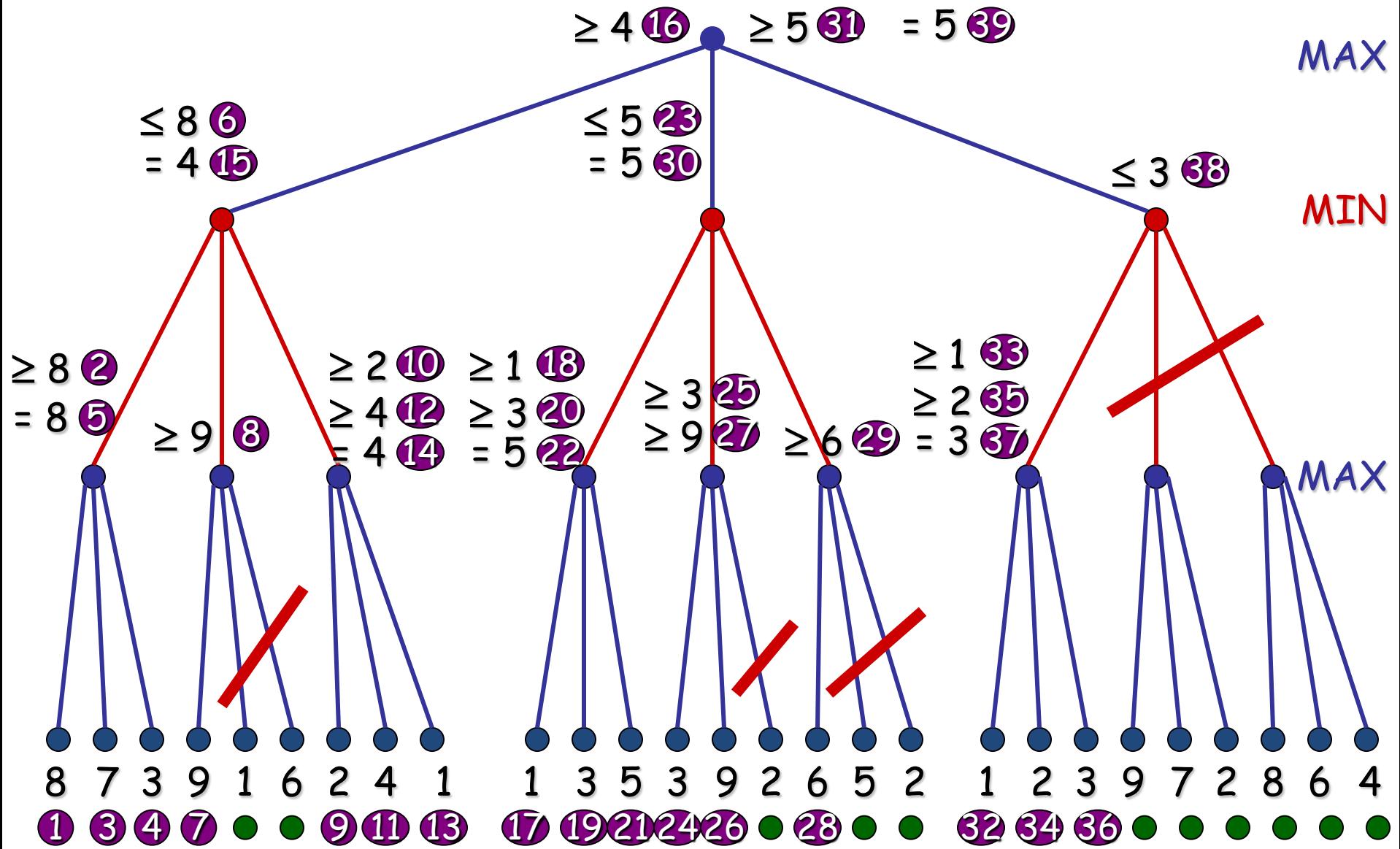
Alpha-Beta Pruning Example



Alpha-Beta Pruning Example



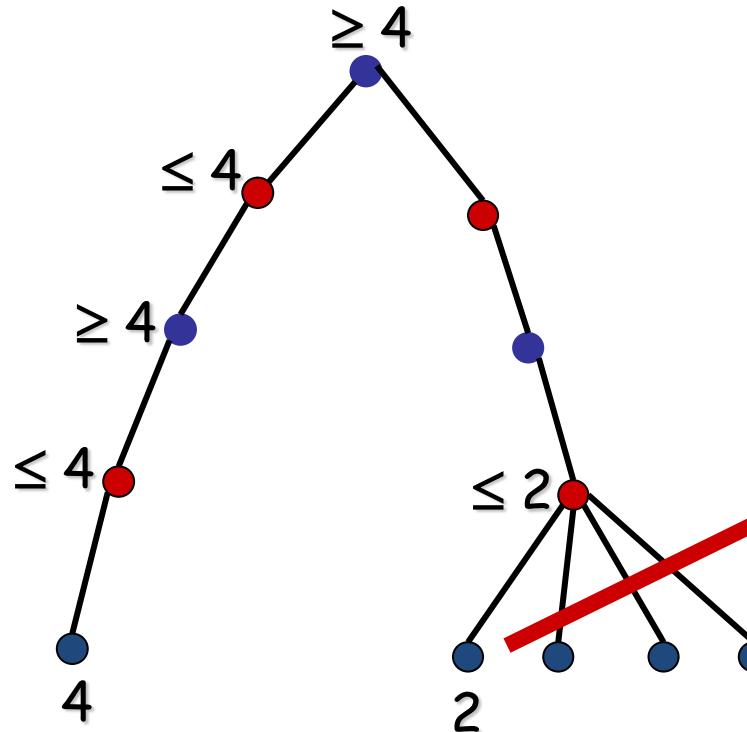
Mini-Max with $\alpha-\beta$ at work:



11 static evaluations saved !!

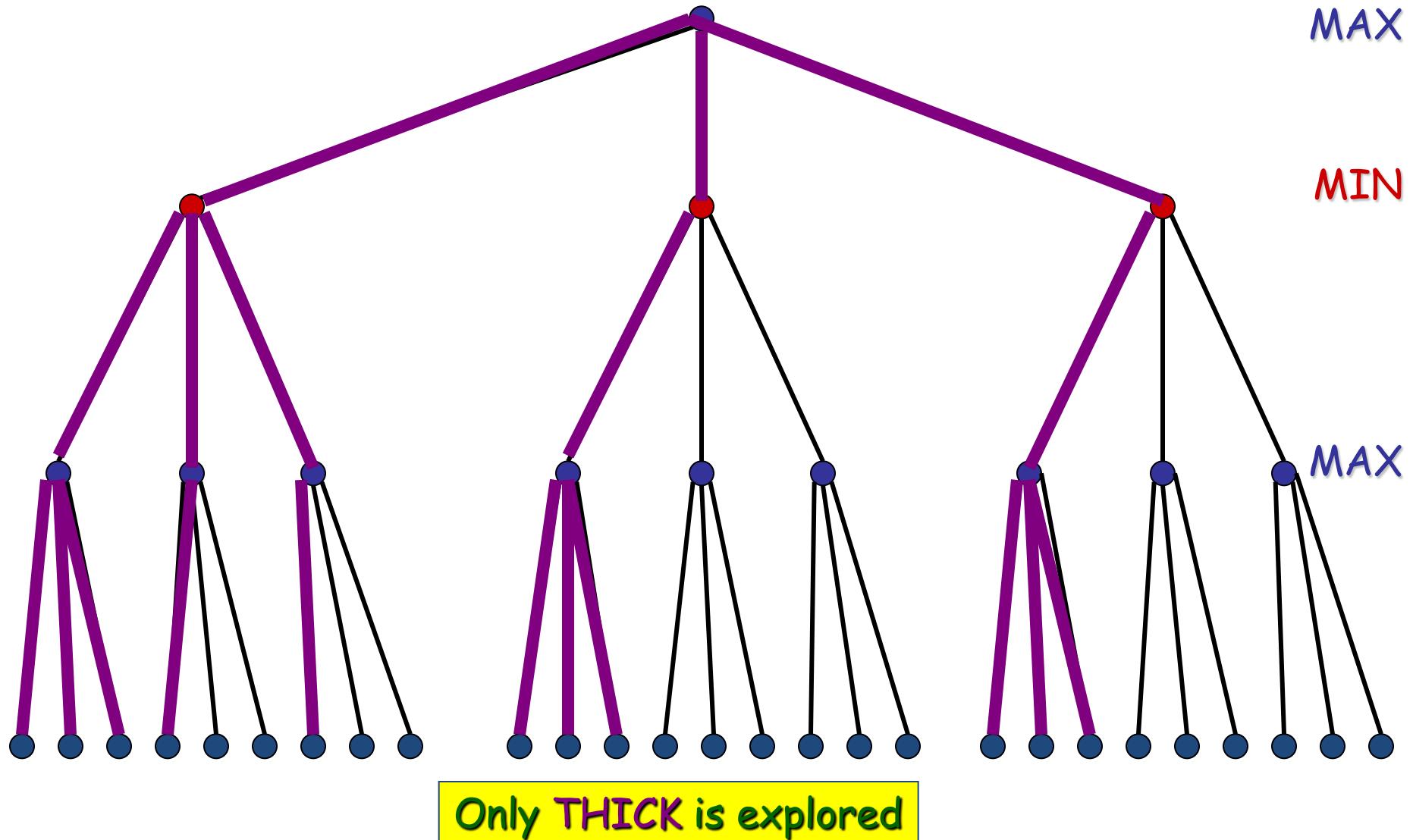
“DEEP” cut-offs

- For game trees with at least 4 Min/Max layers:
the **Alpha - Beta** rules apply also to deeper levels.

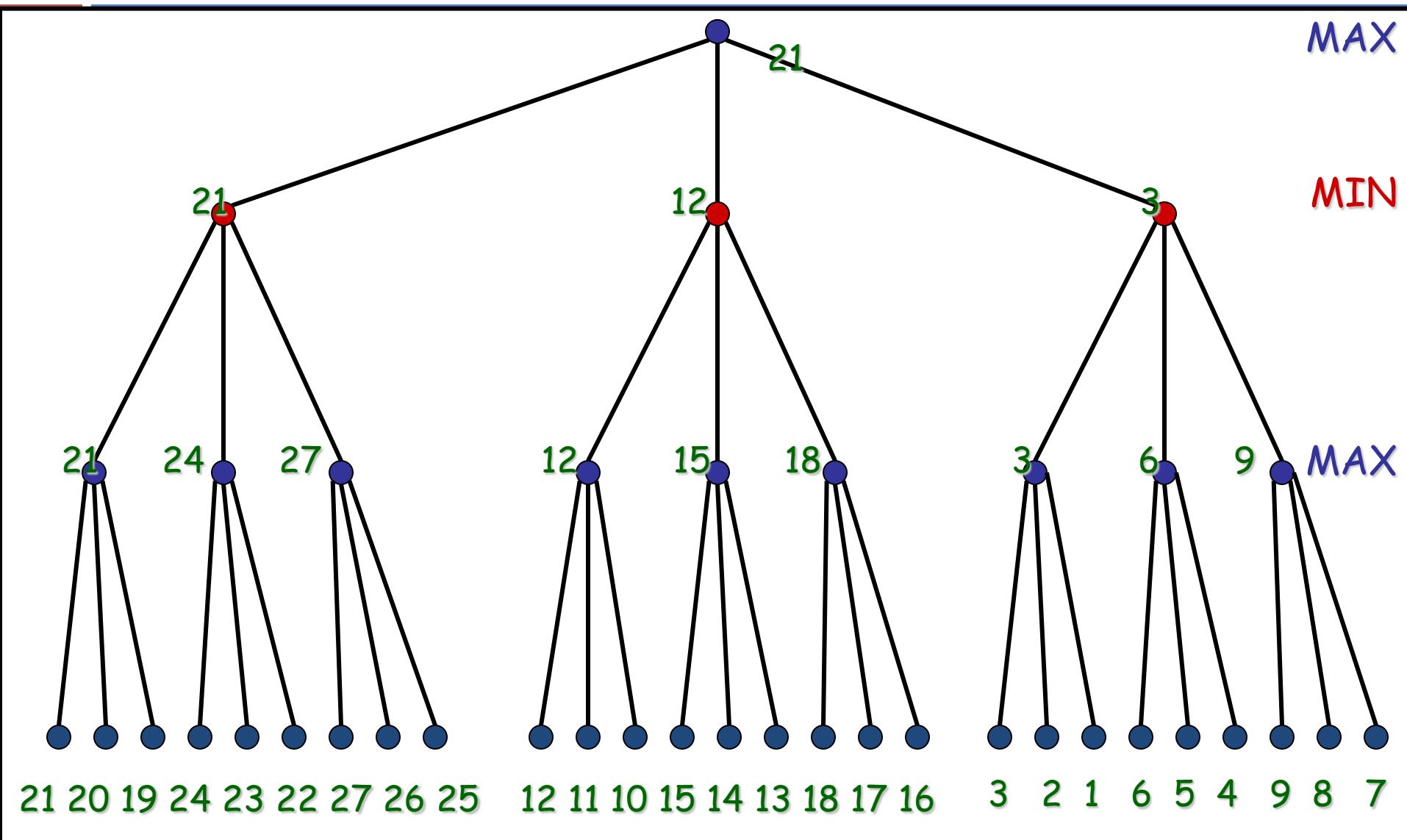


The Gain: Best Case:

- If at every layer: the **best node** is the **left-most one**



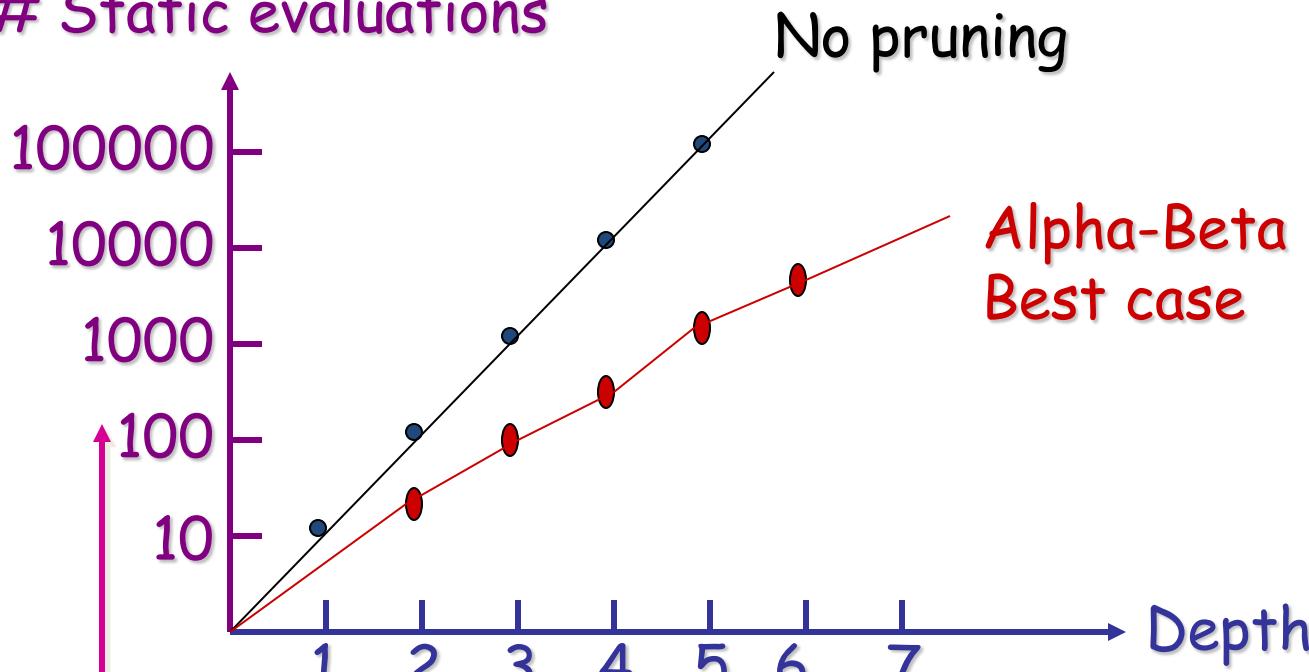
Example of a perfectly ordered tree



Best case gain pictured:

Static evaluations

$b = 10$

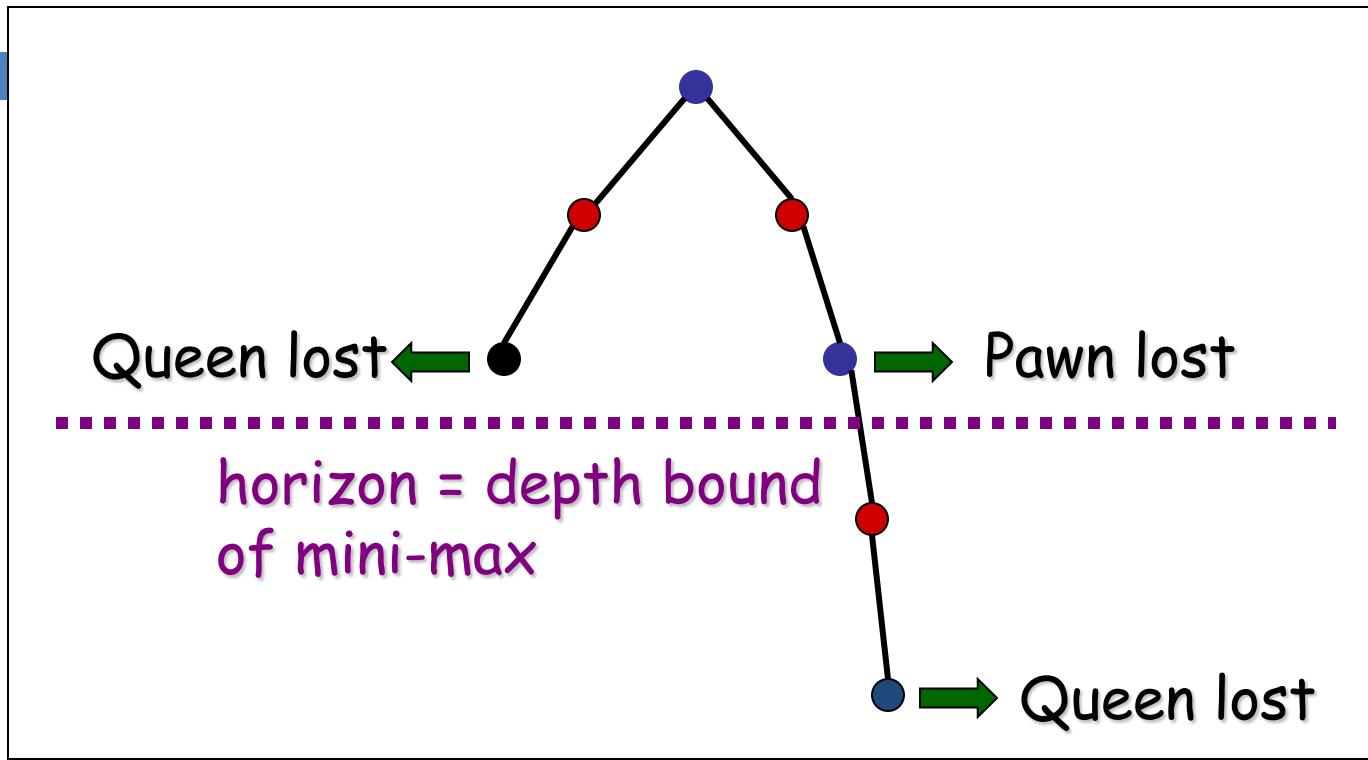


- Note: algorithmic scale.
- Conclusion: still exponential growth !!
- Worst case??

For some trees alpha-beta does nothing,

For some trees: impossible to reorder to avoid cut-offs

The horizon effect



Because of the depth-bound
we prefer to delay disasters, although we don't
prevent them !!

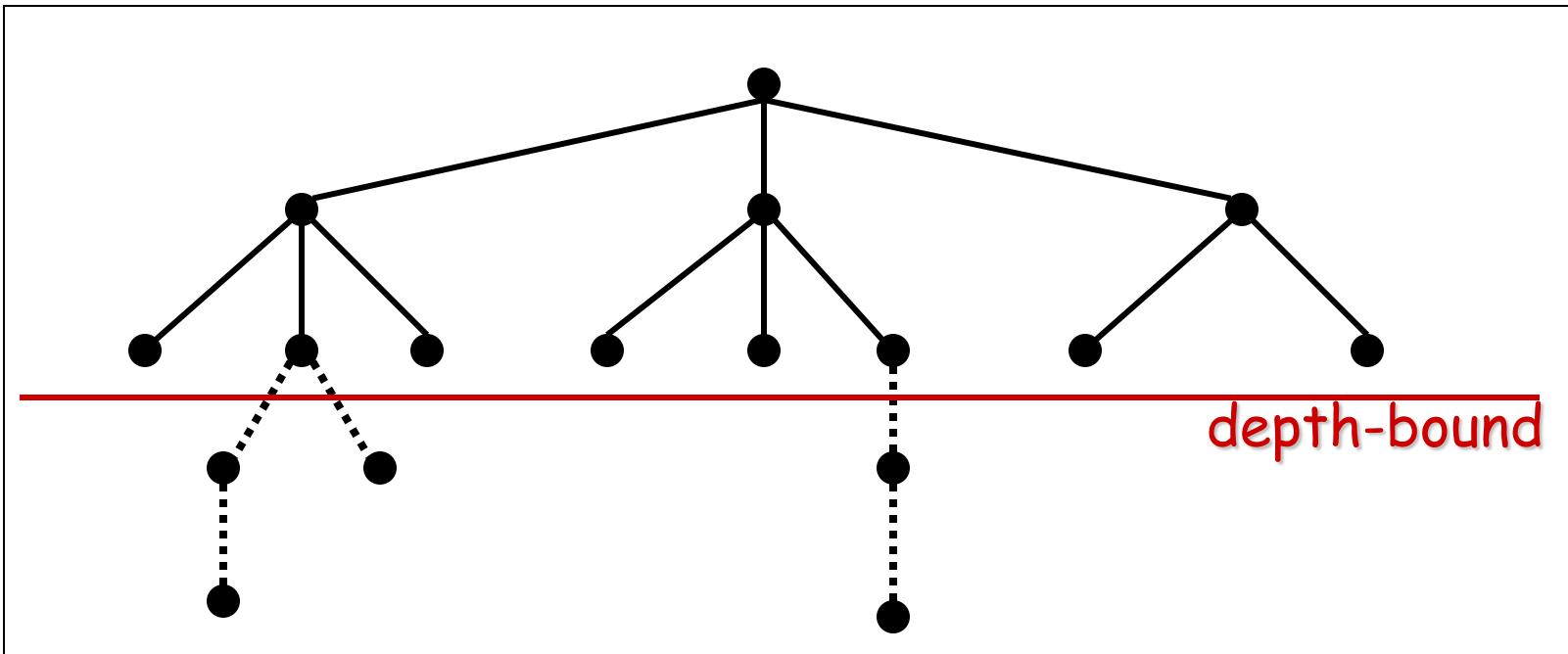
- solution: heuristic continuations

Heuristic Continuation

In situations that are identified as strategically crucial

e.g: king in danger, imminent piece loss, pawn to become a queen, ...

extend the search beyond the depth-bound !



Time bounds:

How to play within reasonable time bounds?

Even with fixed depth-bound, times can vary strongly!

Solution:

Iterative Deepening !!!

Summary

- alpha-beta algorithm does the same calculation as minimax, and is more efficient since it prunes irrelevant branches
- usually, the complete game tree is not expanded, search is cut off at some point and an evaluation function calculated to estimate the utility of a state
- So far, for a possibly good and efficient search:
 - select good search method/technique
 - provide info/heuristic if possible
 - apply prune irrelevant branches

ARTIFICIAL INTELLIGENCE

PART 4 – GAME PLAYING

Njeri Ireri

July – October 2021

We Shall Discuss

- Games. Why?
- Minimax search
- Alpha-beta pruning

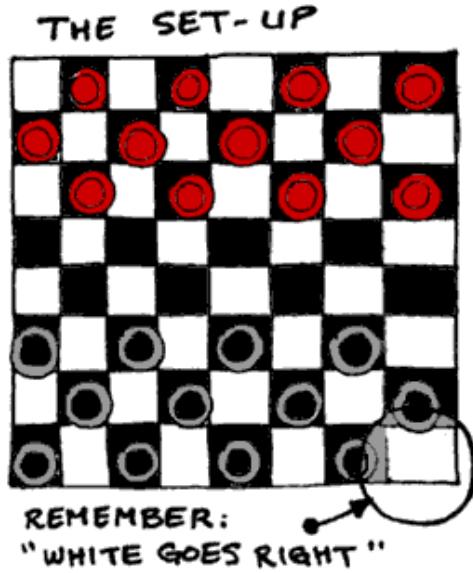
Why are people interested in games?

Some games:

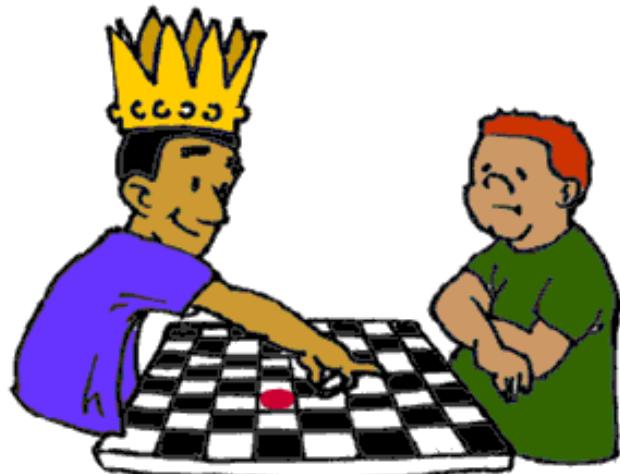
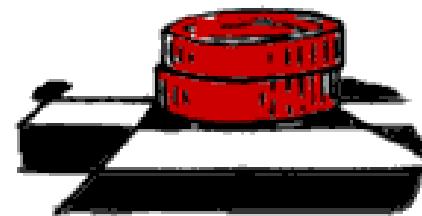
- Ball games
- Card games
- Board games
- Computer games
- ...

Why are people interested in games?

Draughts



"CROWNED"



Is a computer more intelligent
if it beats you in a game of draughts?

Why are people interested in games? Robocup



Why study board games ?

- One of the oldest sub-fields of AI.
- Game Playing has been studied for a long time
 - Babbage (tic-tac-toe)
 - Turing (chess)
- It is an abstract and pure form of competition that seems to require intelligence
- Easy to represent the states and actions
 - Very little world knowledge required !
- Game playing research has contributed ideas on how to make the best use of time to reach good decisions, when reaching optimal decisions is impossible. These ideas are applicable in tackling real-world search problems
 - “A chess (or draughts) playing computer would be proof of a machine doing something thought to require intelligence”

Game playing

- Game playing is a special case of a search problem, with some new requirements
- Up till now we have assumed the situation is not going to change whilst we search
 - static environment but
- Game playing is not like this, because
 - The opponent introduces uncertainty
 - The opponent also wants to win

Our limit here: 2-person games, with no chance

Game playing

- “Contingency” problem:
 - We don’t know the opponents move !
- The size of the search space:
 - Chess : ~15 moves possible per state, 80 ply
 - 15^{80} nodes in tree
 - Go : ~200 moves per state, 300 ply
 - 200^{300} nodes in tree
- Game playing algorithms: they involve
 - Search tree only up to some depth bound
 - Use an evaluation function at the depth bound
 - Propagate the evaluation upwards in the tree

Types of games

	<u>Deterministic</u>	<u>Chance</u>
<u>Perfect information</u>	Chess, draughts, Go, Othello, Tic-tac-toe, Ajua	Backgammon, Monopoly
<u>Imperfect information</u>		Bridge, Poker, Scrabble

Game Playing - Chess

- Shannon - March 9th 1949 - New York
- Size of search space (10^{120} - average of 40 moves)
- 200 million positions/second = 10^{100} years to evaluate all possible games
- Searching to depth = 40, at one node per microsecond it would take 10^{90} years to make its first move

Game Playing - Chess

- 1957 - Newell and Simon predicted that a computer would be chess champion within ten years
- Simon : “I was a little far-sighted with chess, but there was no way to do it with machines that were as slow as the ones way back then”
- 1958 - First computer to play chess was an IBM 704 - about one millionth capacity of deep blue.
- 1967 : Mac Hack competed successfully in human tournaments
- 1983 : “Belle” obtained expert status from the United States Chess Federation
- Mid 80’s : Scientists at Carnegie Mellon University started work on what was to become Deep Blue.
- Project moved to IBM in 1989

Game Playing - Chess

- May 11th 1997, Gary Kasparov lost a six match game to Deep blue
 - 3.5 to 2.5
 - Two wins for deep blue, one win for Kasparov and three draws

Game Playing - Checkers (Draughts)

- Arthur Samuel - 1952
- Written for an IBM 701
- 1954 - Re-wrote for an IBM 704
 - 10,000 words of main memory
- Added a learning mechanism that learnt its own evaluation function
- Learnt the evaluation function by playing against itself
- After a few days it could beat its creator
- And compete on equal terms with strong human players
- Jonathon Schaeffer - 1996
- Developed Chinook

Game Playing - Checkers (Draughts)

- Chinook
- Uses Alpha-Beta search
- Plays a perfect end game by means of a database
- In 1992 Chinook won the US Open
- And challenged for the world championship
- Dr Marion Tinsley, had been world championship for over 40 years
 - ... only losing three games in all that time
 - Against Chinook she suffered her fourth and fifth defeat
 - But ultimately won 21.5 to 18.5

Game Playing - Checkers (Draughts)

- In August 1994 there was a re-match but Marion Tinsley withdrew for health reasons
- Chinook became the official world champion
- Schaeffer claimed Chinook was rated at 2814
- The best human players are rated at 2632 and 2625
- Chinook did not include any learning mechanism

Game Playing - Checkers (Draughts)

- Kumar - 2000
- “Learnt” how to play a good game of checkers
- The program used a *population* of games with the best competing for *survival*
- Learning was done using a *neural network* with the synapses being changed by an *evolutionary strategy*
- The best program beat a commercial application 6-0

- The program was presented at CEC 2000 (San Diego) and remain undefeated

Game Playing - Minimax

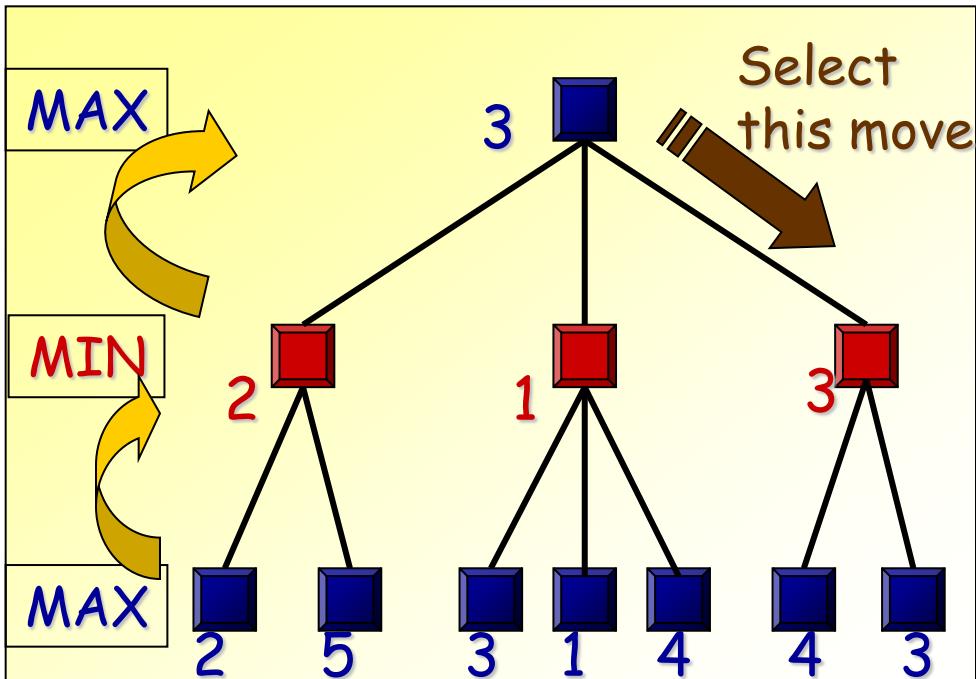
- Game Playing: An opponent tries to thwart your every move
- 1944 - John von Neumann outlined a search method (**Minimax**) that maximised your position whilst minimising your opponents
- In order to implement we need a method of measuring how good a position is
 - Often called a **utility function** (or **payoff function**)
 - e.g. outcome of a game; win 1, loss -1, draw 0
- Initially this will be a value that describes our position exactly

Game Playing - Minimax

- The MiniMax algorithm **selects the “best” next move for a computer player in a two-player game.** The algorithm makes a tree of all possible moves for both players
- This algorithm is called MiniMax simply because the computer makes moves that bring it **maximum gain**, while assuming the opponent makes moves that brings the computer **minimum gain**. Because the players alternate moves, the algorithm alternates between minimizing and maximizing levels of the recursive search tree
- Let's look at a hypothetical search tree to see how the MiniMax analysis works to ultimately select the best move; this example shows a game where there are either two or three possible moves, and where the look ahead limit (search depth) is two moves...

Game Playing - Minimax

- Restrictions:
 - 2 players: MAX (computer) and MIN (opponent)
 - deterministic, perfect information
- Select a depth-bound (say: 2) and evaluation function



- Construct the tree up till the depth-bound
- Compute the evaluation function for the leaves
- Propagate the evaluation function upwards:
 - taking minima in MIN
 - taking maxima in MAX

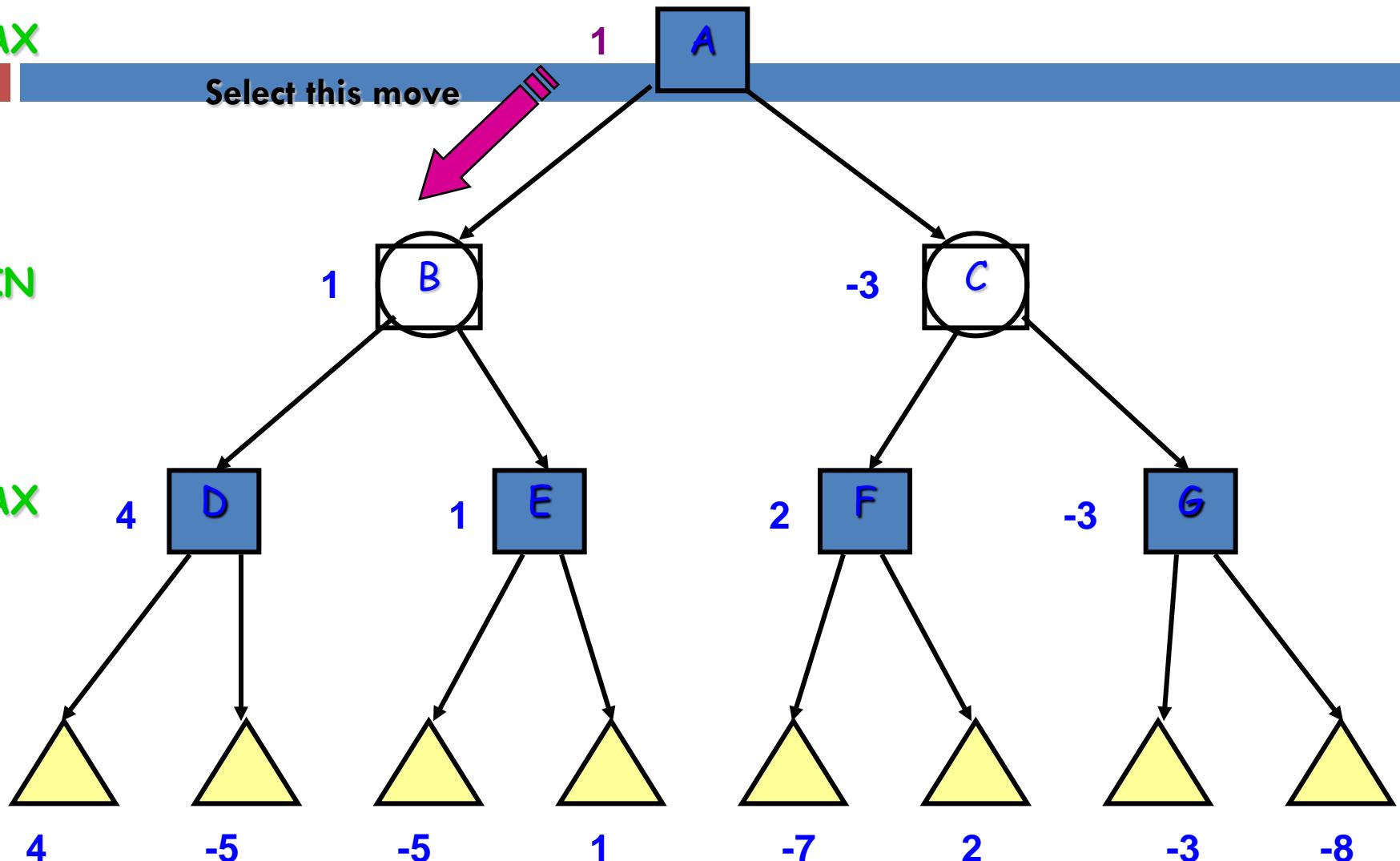
Game Playing - Minimax Example

MAX

Select this move

MIN

MAX



= terminal position



= agent



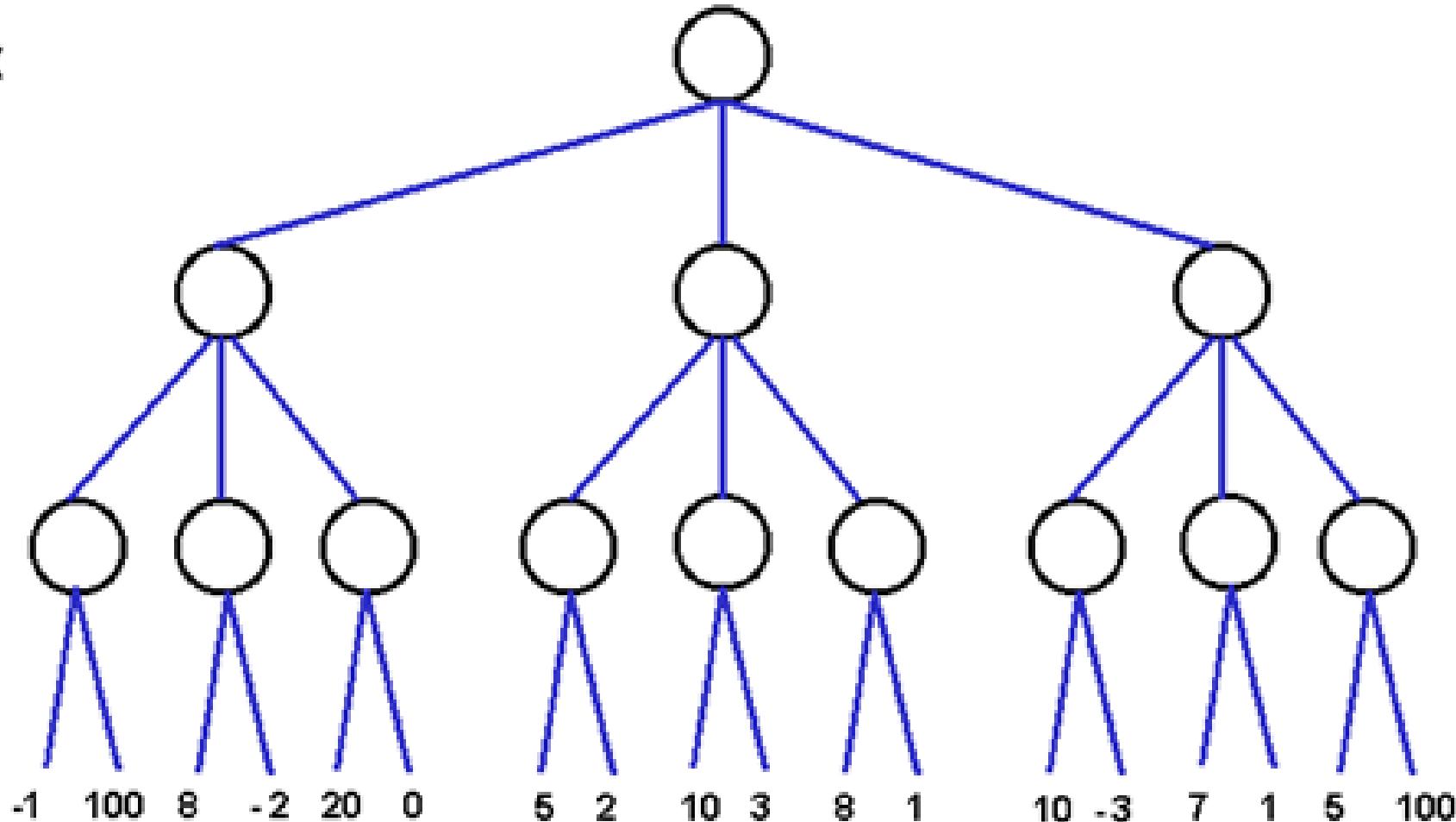
= opponent

Exercise

MAX

MIN

MAX



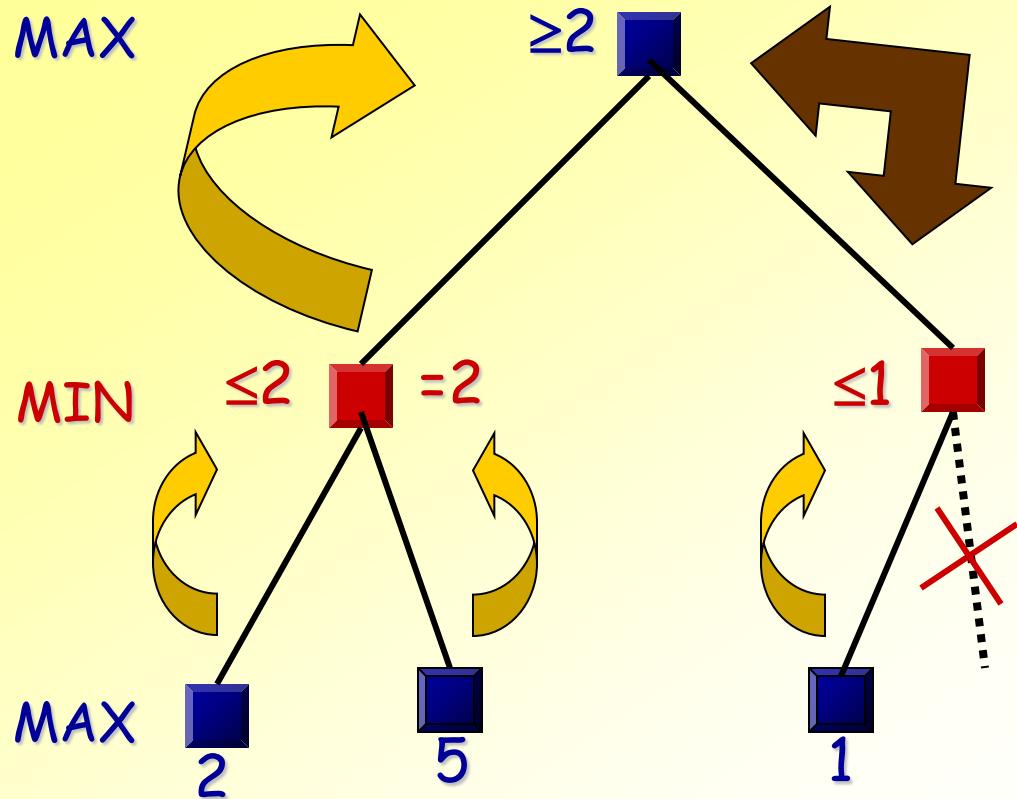
Game Playing - Alpha-Beta Pruning

- Generally applied optimization on Mini-max
- Instead of:
 - first creating the entire tree (up to depth-level)
 - then doing all propagation
- Interleave the generation of the tree and the propagation of values
- Point:
 - some of the obtained values in the tree will provide information that other (non-generated) parts are redundant and do not need to be generated

Alpha-Beta idea:

□ Principles:

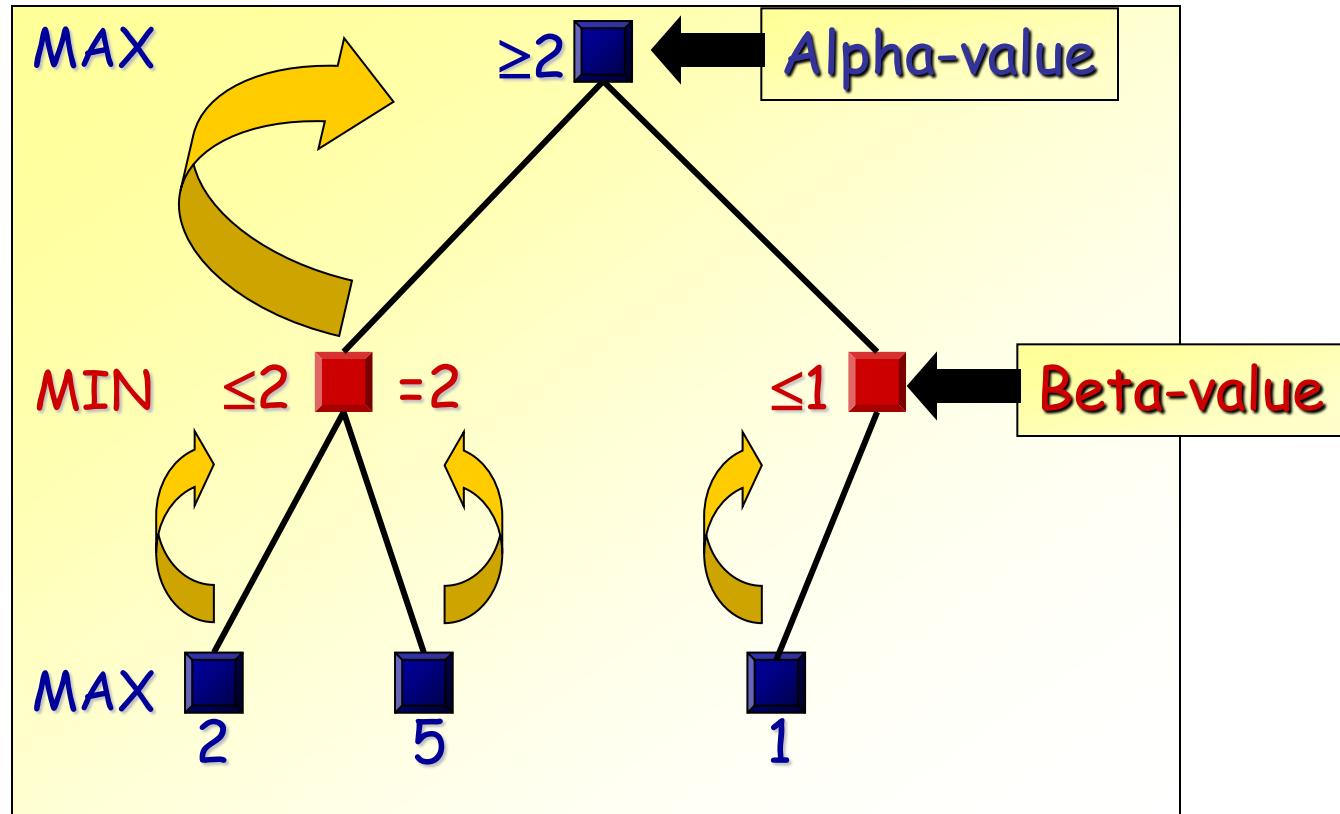
- generate the tree depth-first, left-to-right
- propagate final values of nodes as initial estimates for their parent node



- The **MIN**-value (1) is already smaller than the **MAX**-value of the parent (2)
- The **MIN**-value can only decrease further,
- The **MAX**-value is only allowed to increase,
- No point in computing further below this node

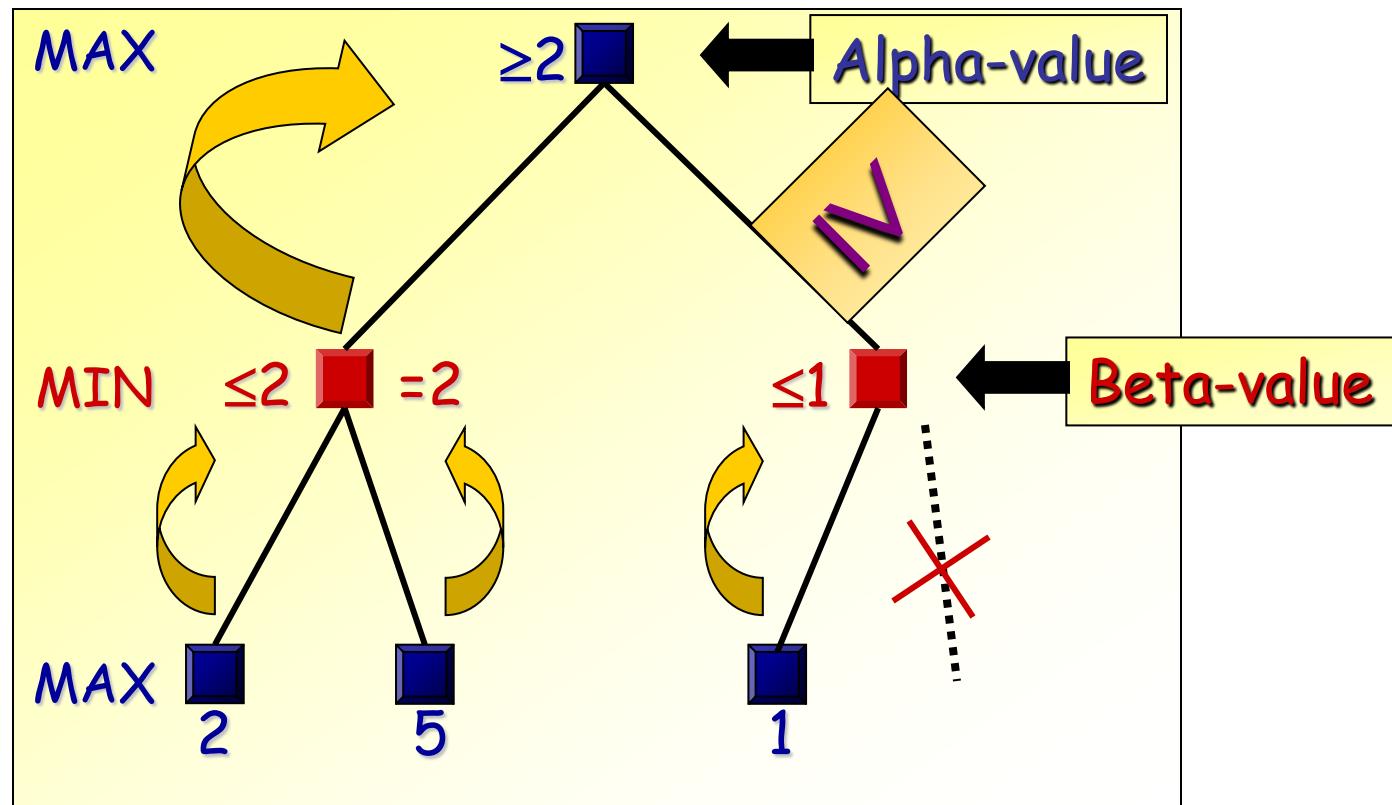
Terminology:

- The (temporary) values at **MAX**-nodes are **ALPHA**-values
- The (temporary) values at **MIN**-nodes are **BETA**-values



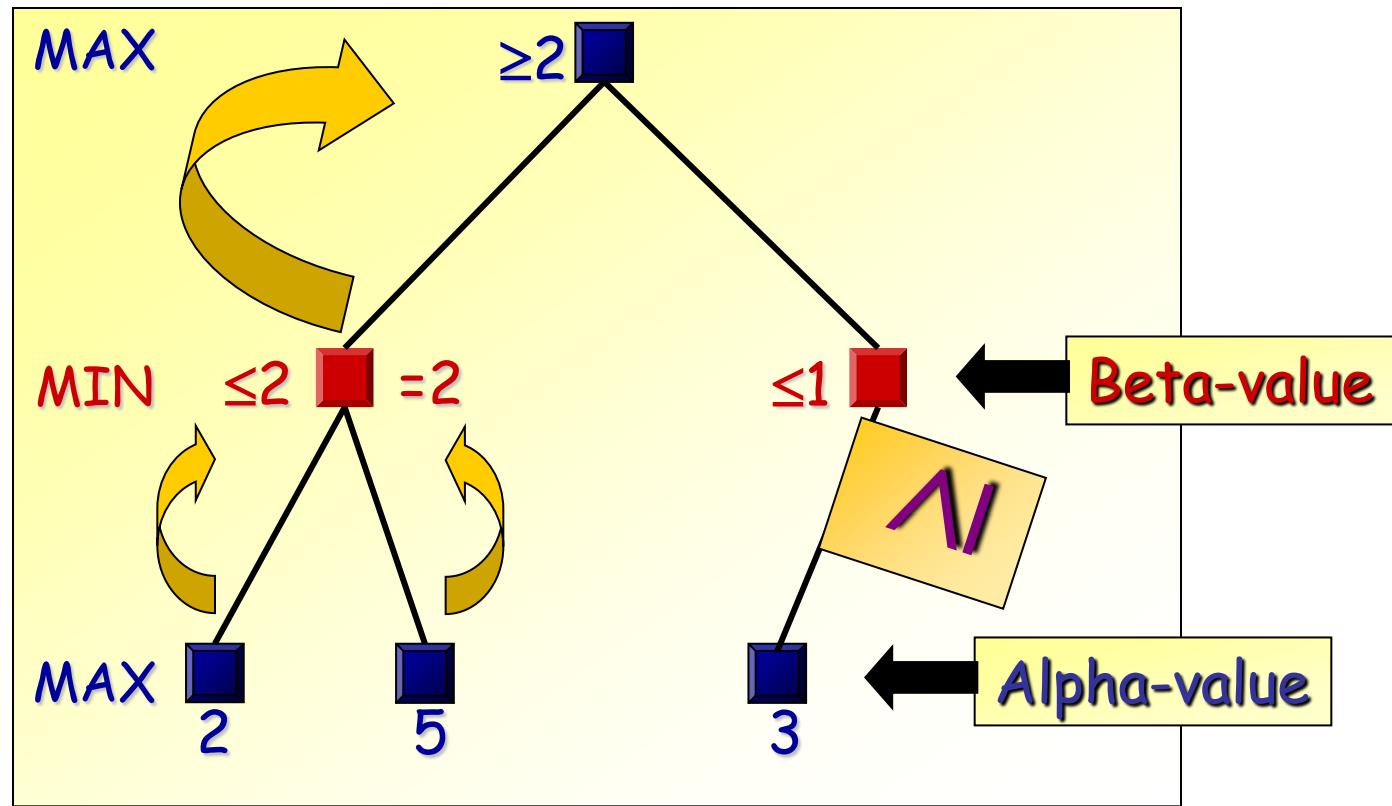
The Alpha-Beta principles (1):

- If an ALPHA-value is larger or equal than the Beta-value of a descendant node:
stop generation of the children of the descendant

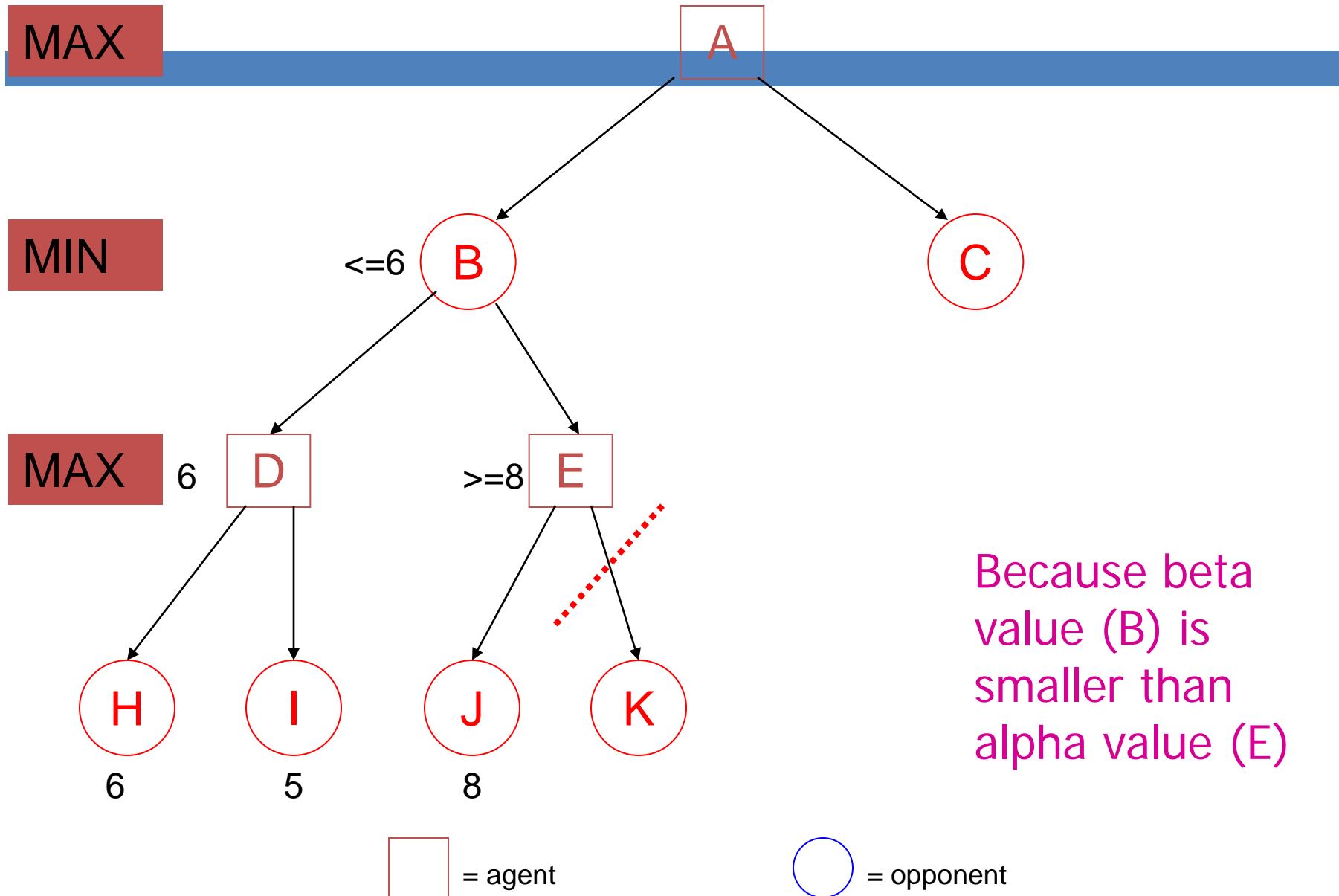


The Alpha-Beta principles (2):

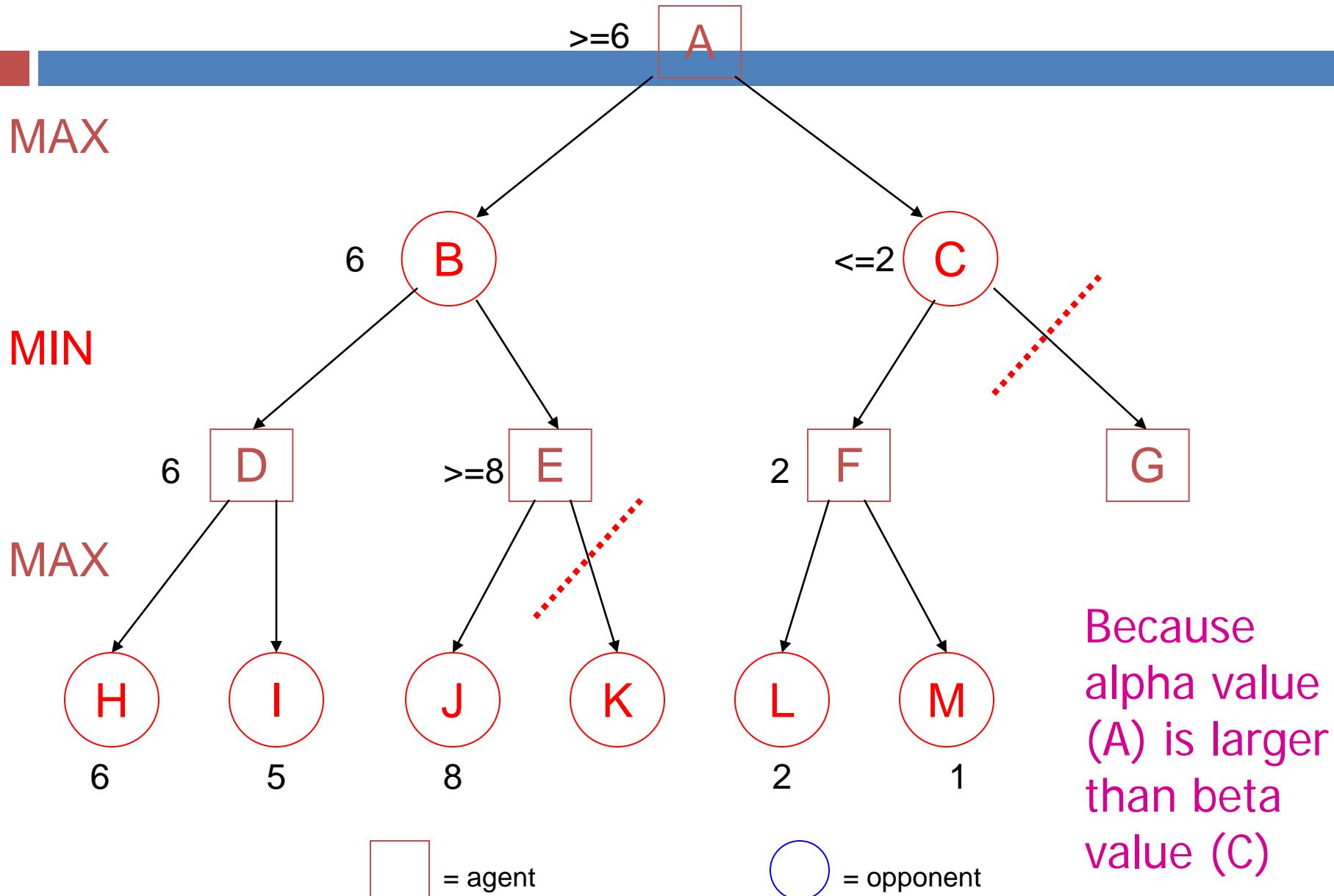
- If an **Beta-value** is **smaller or equal** than the **Alpha-value** of a descendant node:
stop generation of the children of the descendant



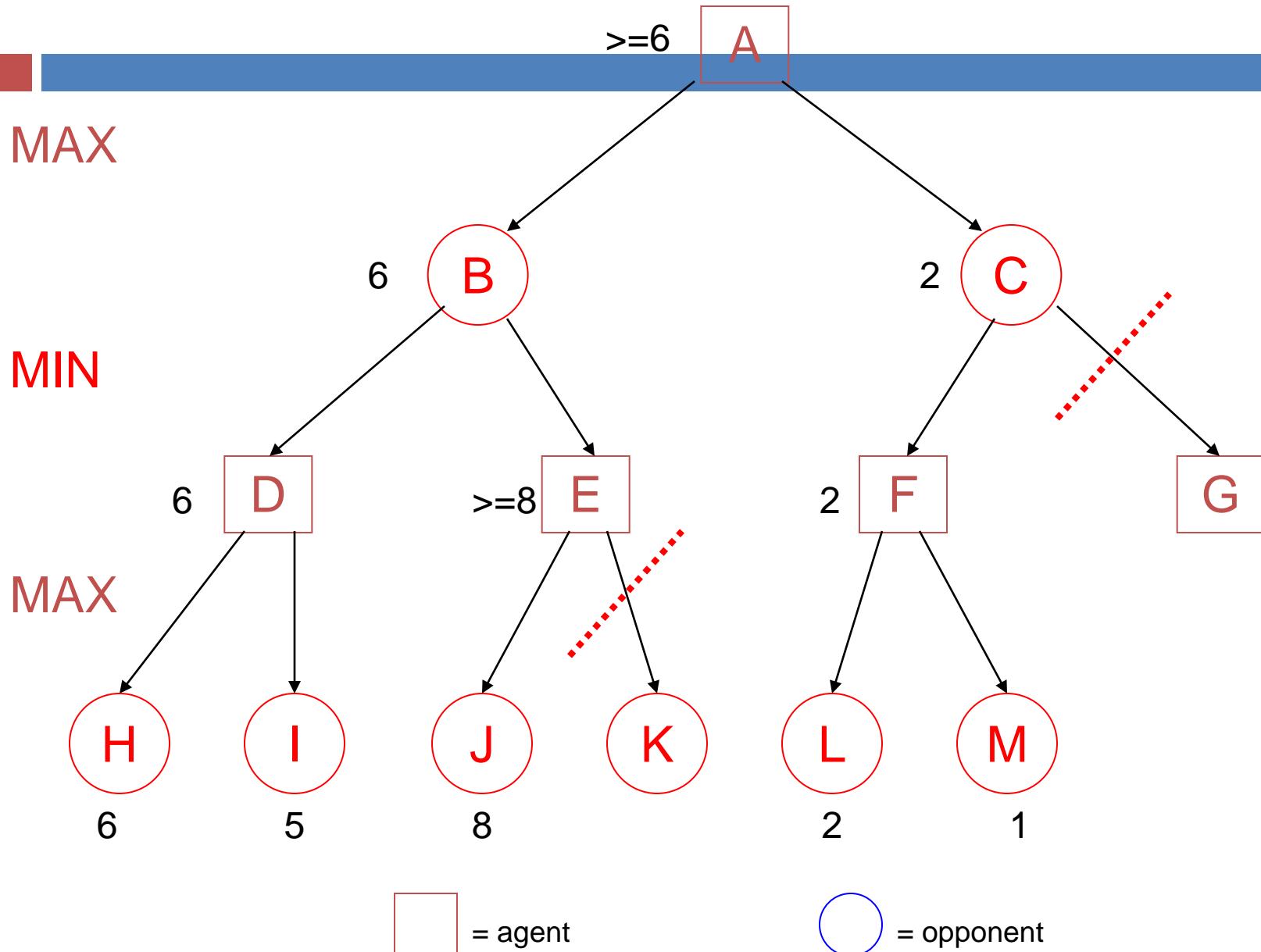
Alpha-Beta Pruning Example



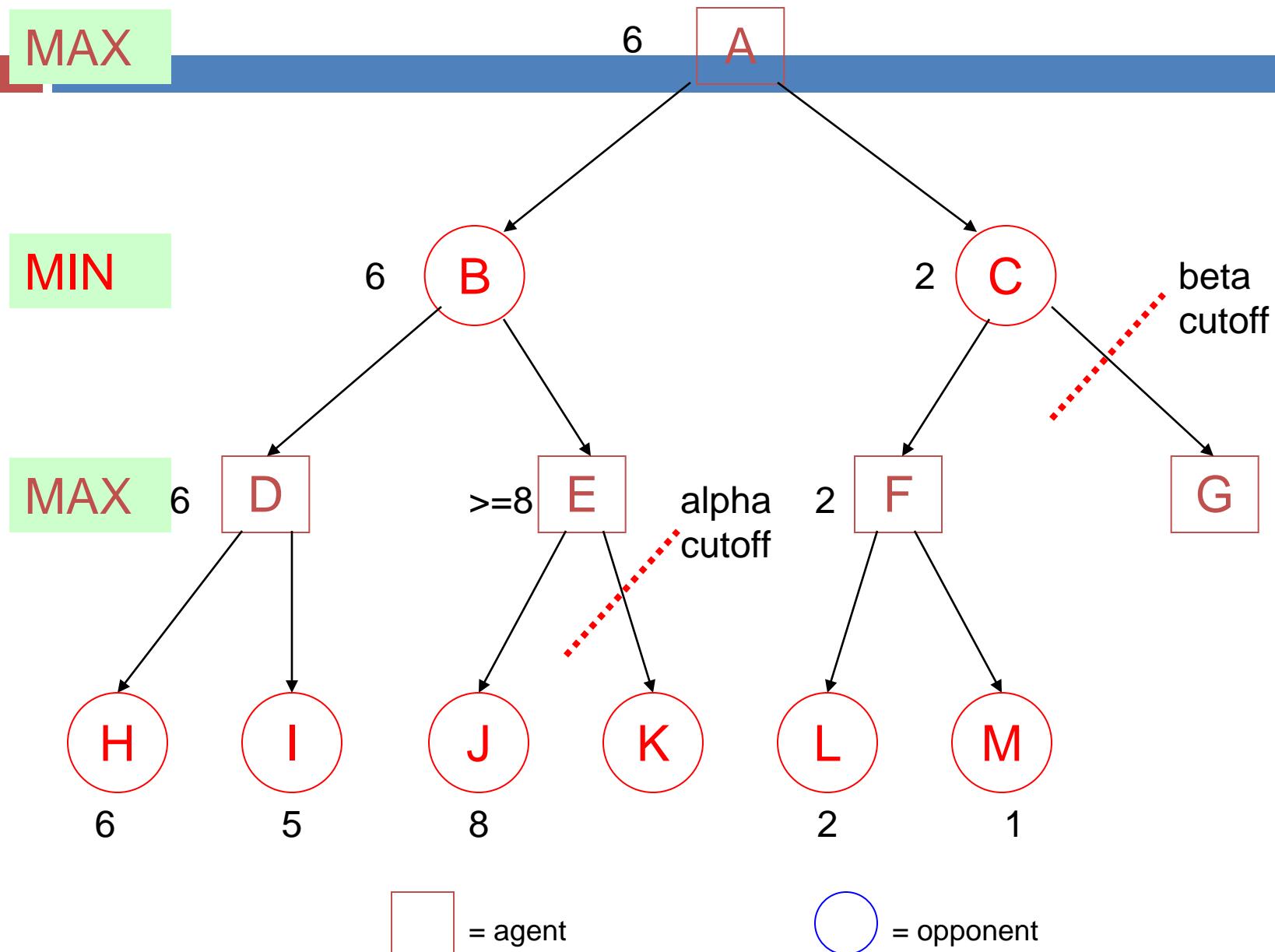
Alpha-Beta Pruning Example



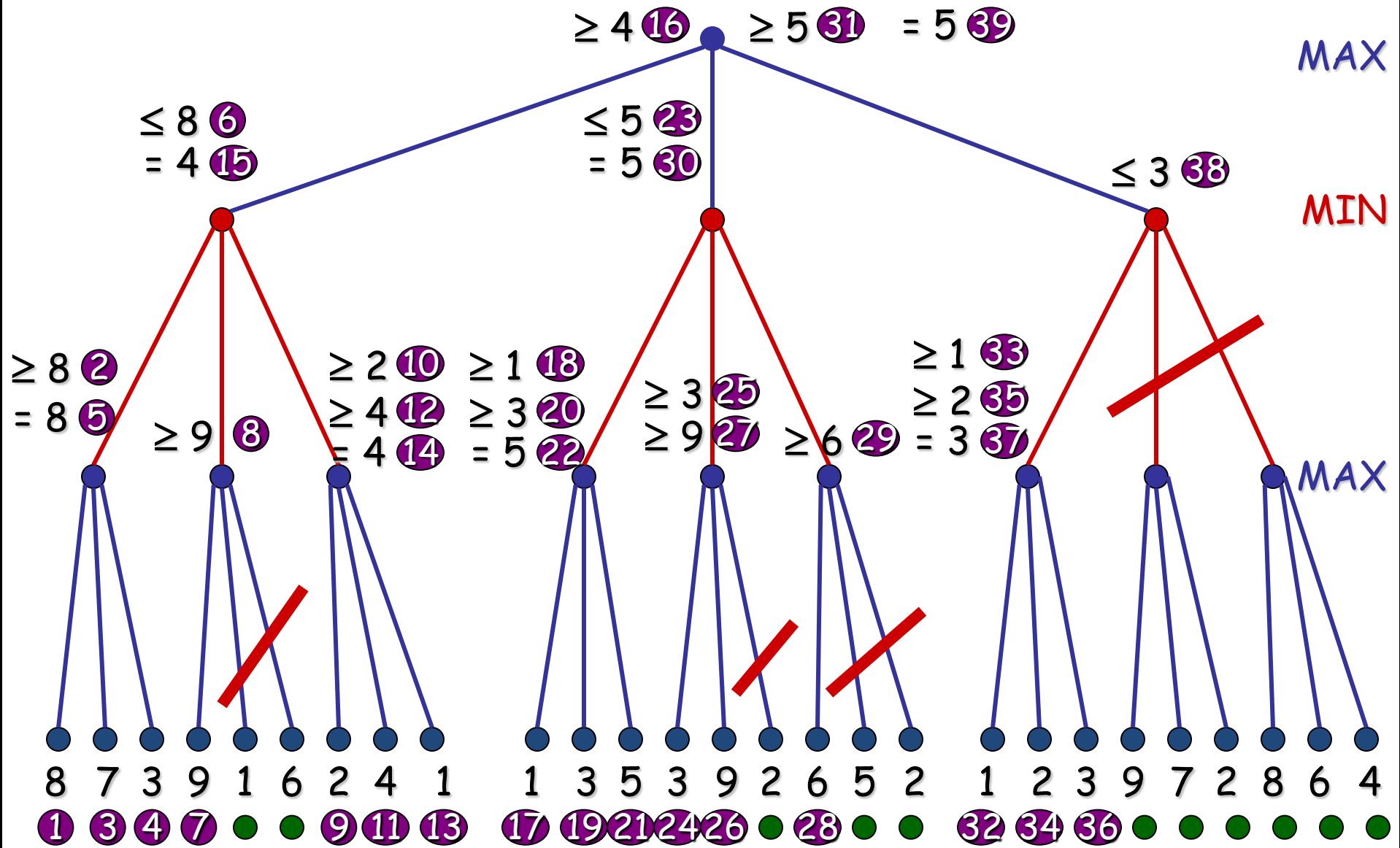
Alpha-Beta Pruning Example



Alpha-Beta Pruning Example



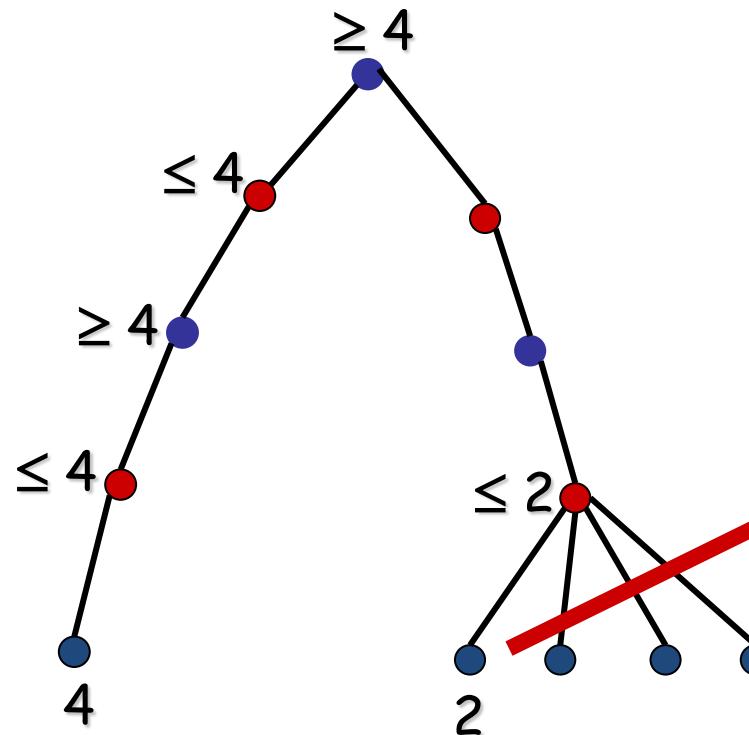
Mini-Max with α - β at work:



11 static evaluations saved !!

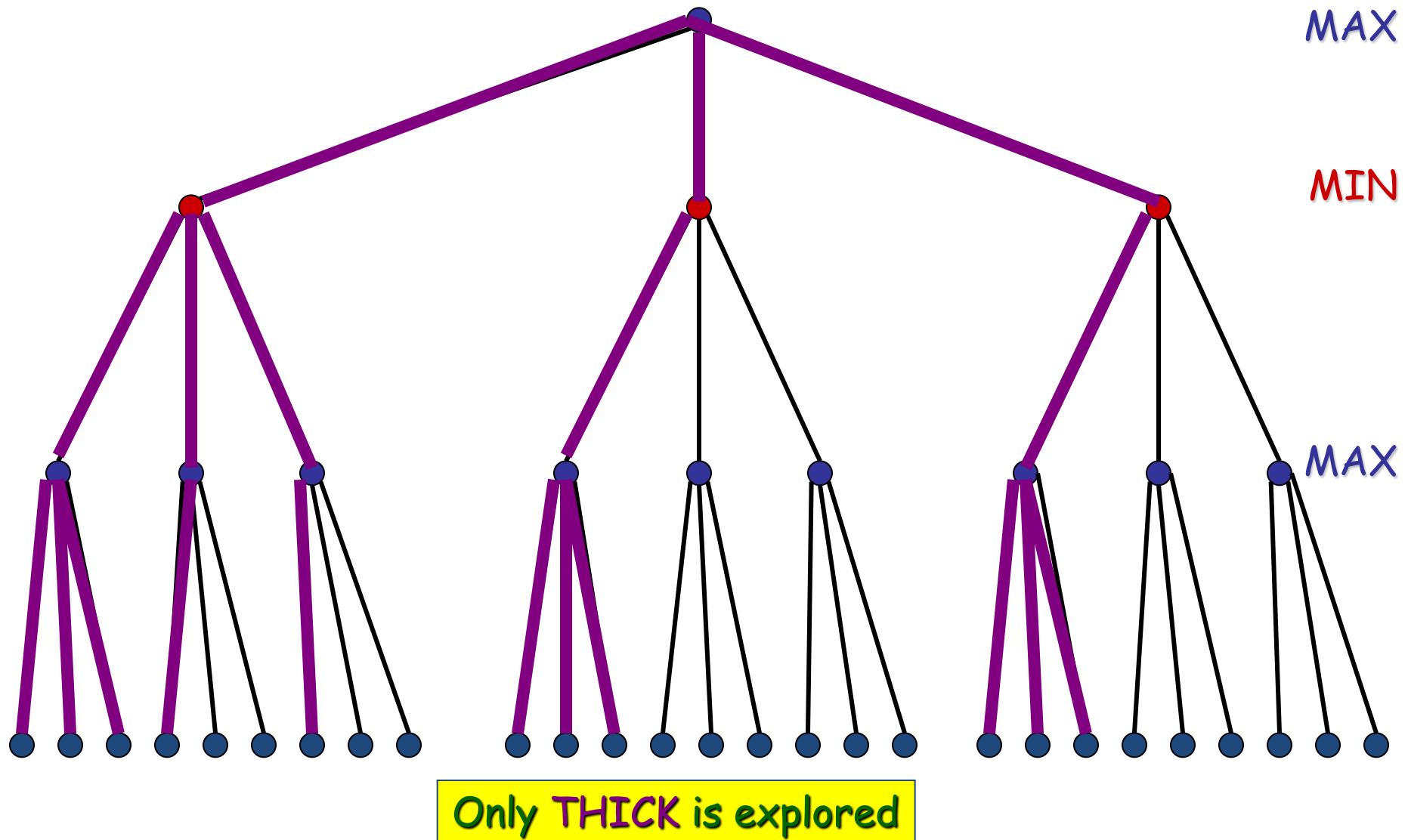
“DEEP” cut-offs

- For game trees with at least 4 **Min/Max** layers:
the **Alpha - Beta** rules apply also to deeper levels.

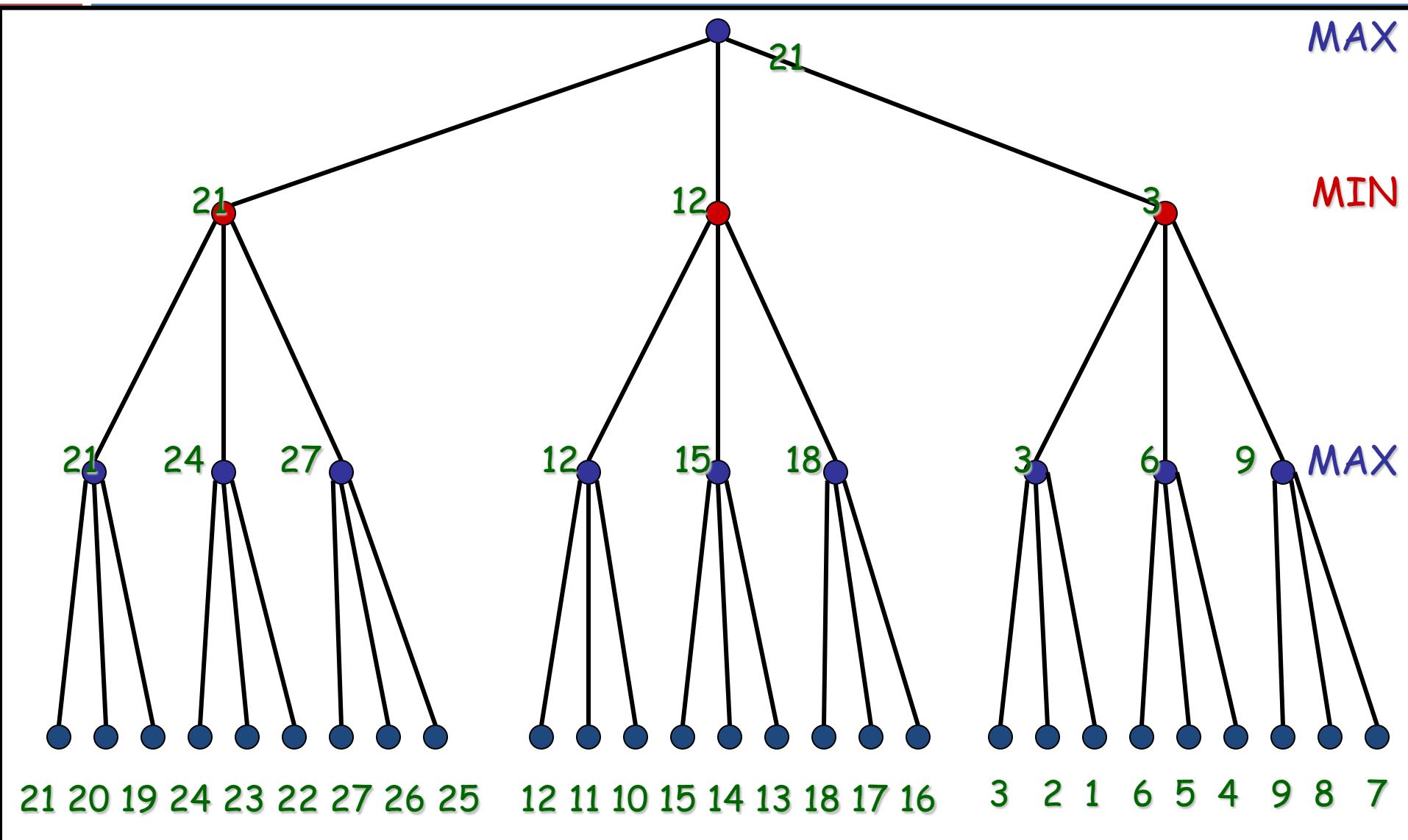


The Gain: Best Case:

- If at every layer: the **best node** is the **left-most one**



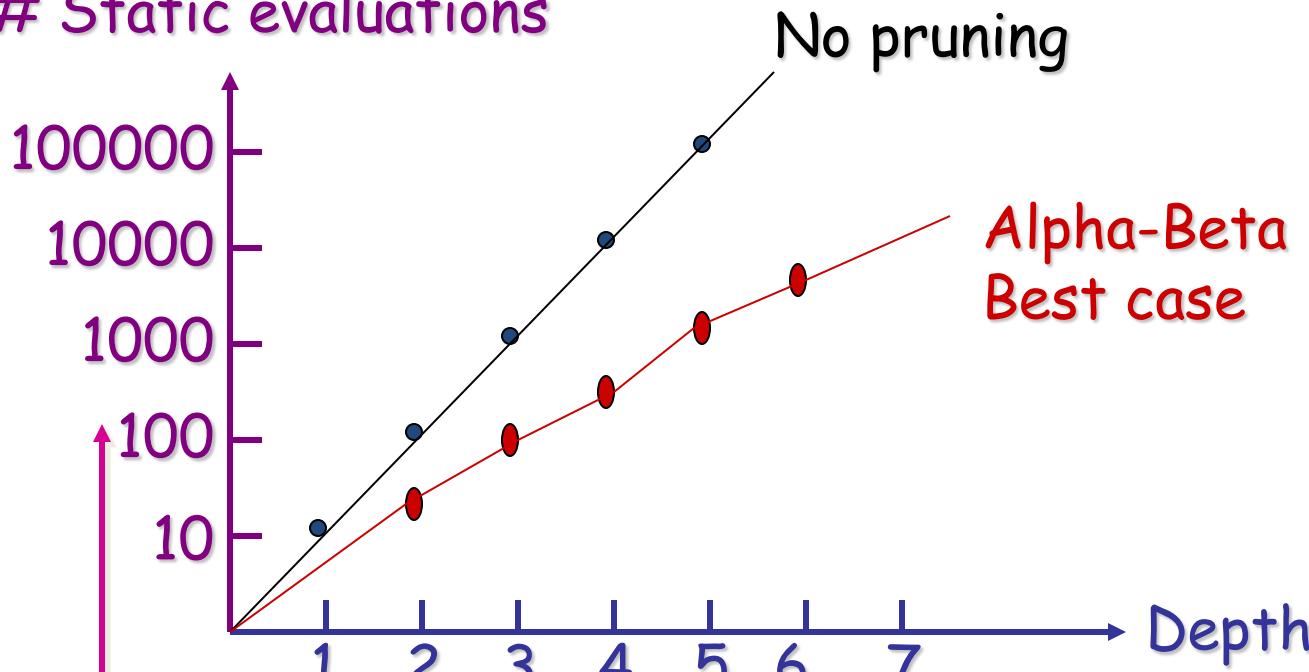
Example of a perfectly ordered tree



Best case gain pictured:

Static evaluations

$b = 10$

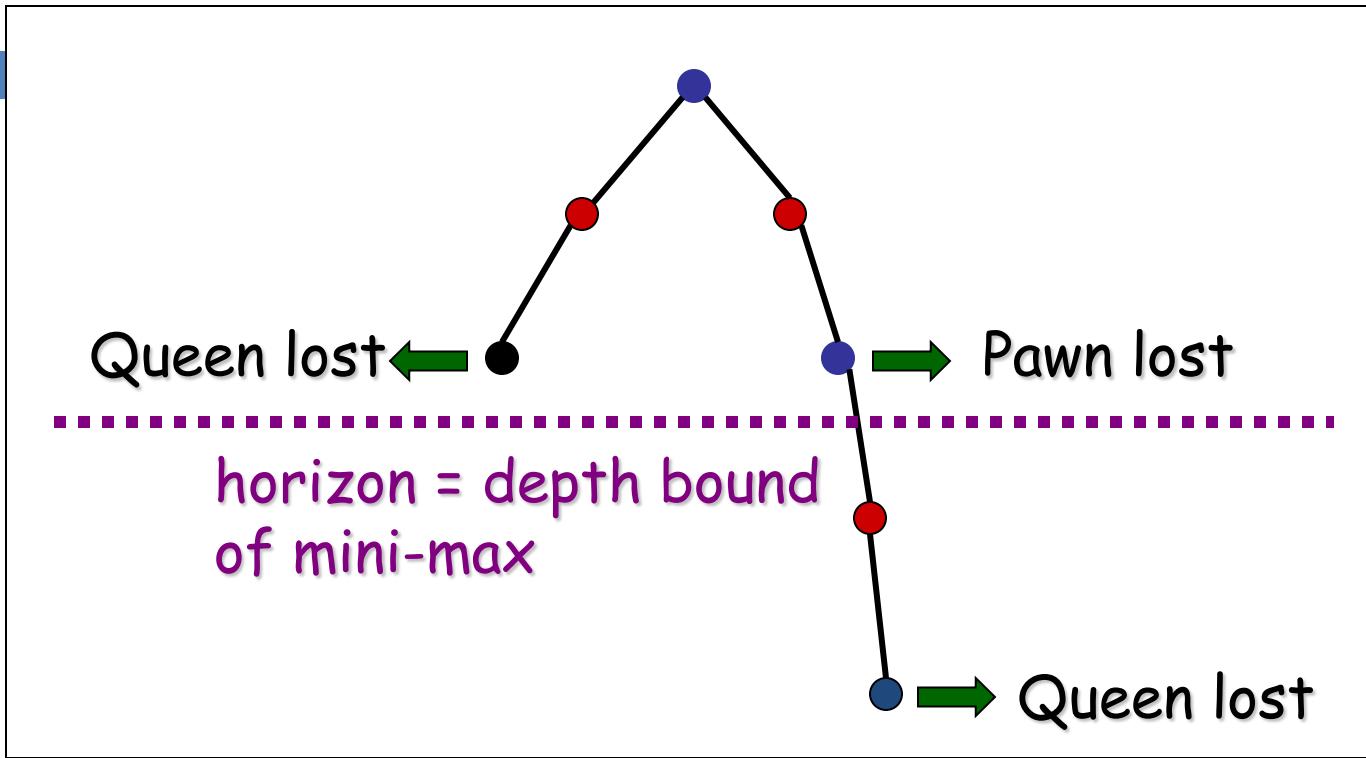


- Note: algorithmic scale.
- Conclusion: still exponential growth !!
- Worst case??

For some trees alpha-beta does nothing,

For some trees: impossible to reorder to avoid cut-offs

The horizon effect



Because of the depth-bound
we prefer to delay disasters, although we don't
prevent them !!

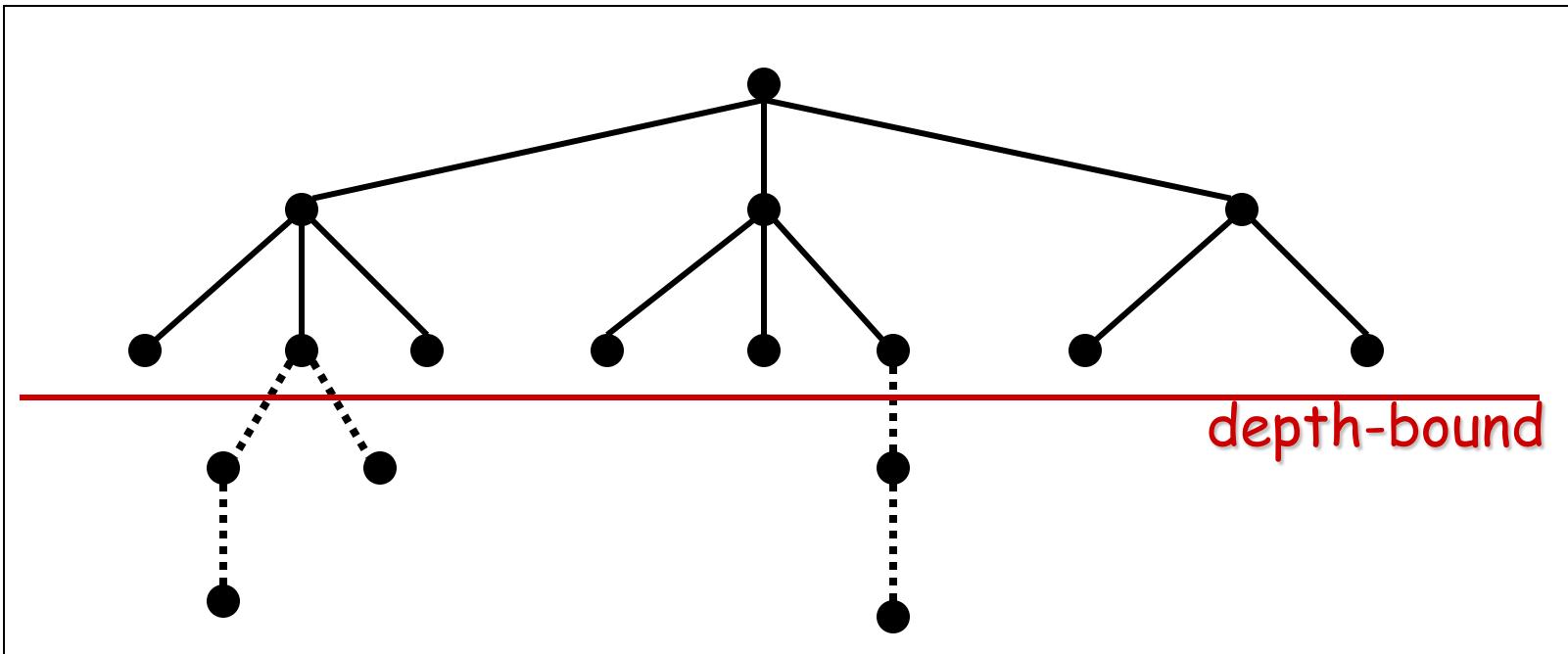
- solution: heuristic continuations

Heuristic Continuation

In situations that are identified as strategically crucial

e.g: king in danger, imminent piece loss, pawn to become a queen, ...

extend the search beyond the depth-bound !



Time bounds:

How to play within reasonable time bounds?

Even with fixed depth-bound, times can vary strongly!

Solution:

Iterative Deepening !!!

Summary

- alpha-beta algorithm does the same calculation as minimax, and is more efficient since it prunes irrelevant branches
- usually, the complete game tree is not expanded, search is cut off at some point and an evaluation function calculated to estimate the utility of a state
- So far, for a possibly good and efficient search:
 - select good search method/technique
 - provide info/heuristic if possible
 - apply prune irrelevant branches

Evolutionary Algorithms (Goal: get the “optimal” solution)

Many AI applications have a search space that is exponentially proportional to the problem dimensions and for the simplest of cases these problems cannot be solved using exhaustive search methods. Consequently, there is considerable interest in heuristic techniques that attempt to discover near-optimal solutions within an acceptable time.

In AI, an evolutionary algorithm (EA) is a generic population-based meta-heuristic optimization algorithm.

Simply put, EAs are computer programs that attempt to solve complex problems by mimicking the processes of Darwinian evolution. An EA uses mechanisms inspired by biological evolution / natural selection, such as inheritance, Reproduction (crossover and mutation), recombination, and selection.

EAs include the following search and optimization methods:

- Genetic Algorithms
- Evolutionary Programming
- Genetic Programming
- Evolution Strategies
- Classifier systems

Optimization design for a method is based on its components. The most important components in a EA method consist of:

- genetic representation of candidate solutions (definition of individuals)
- evaluation function (fitness function)
- population
- selection scheme
 - parent selection mechanism
 - survivor selection mechanism (replacement)
- Genetic/variation operators (crossover, mutation,...)

Genetic Algorithms in Search and Optimization (Goal: achieve the global (or near global) optimum)

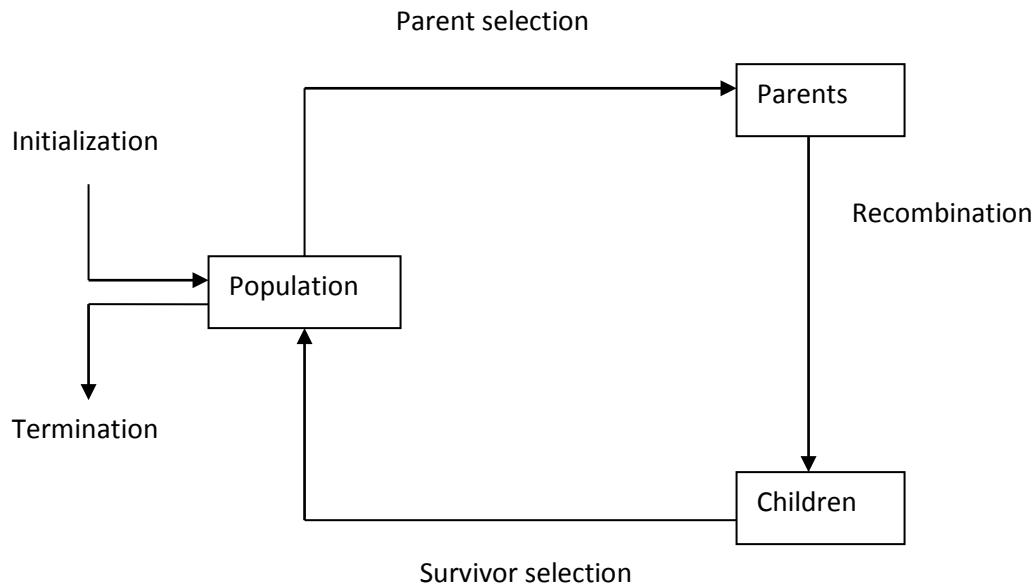
Genetic algorithms (GA) are the most frequently encountered type of evolutionary algorithm. A GA is an adaptive search heuristic that mimics the process of natural selection.

GAs provide a framework for effectively sampling large search spaces, and the basic technique is both broadly applicable and easily tailored to specific problems.

The general scheme of a GA can be given as follows:

```
begin
    INITIALIZE population with random candidate solutions;
    EVALUATE each candidate;
repeat
    SELECT parents;
    RECOMBINE pairs of parents;
    MUTATE the resulting children;
    EVALUATE children;
    SELECT individuals for the next generation
until TERMINATION-CONDITION is satisfied
end
```

The GA can be represented in form of a diagram:



Remember: EAs use mechanisms inspired by biological evolution, such as inheritance, Reproduction (crossover and mutation), recombination, and selection.

Biological Terminology:

- gene
 - functional entity that codes for a specific feature e.g. eye color

- set of possible alleles
- allele
 - value of a gene e.g. blue, green, brown
 - codes for a specific variation of the gene/feature
- locus
 - position of a gene on the chromosome
- genome
 - set of all genes that define a species
 - the genome of a specific individual is called genotype
 - the genome of a living organism is composed of several chromosomes
- population
 - set of competing genomes/individuals

Reproduction methods:

There are two basic methods of reproduction:

- Mutation
- Crossover

Genetic algorithms use crossover (crossover in genetics hence the 'gene' in their name) and mutation to search the space of possible solutions. If only mutation is used, the algorithm is very slow. Crossover makes the algorithm significantly faster.

GA is a kind of **hill-climbing** search. As with all hill-climbing algorithms, there is a problem of local maxima. Local maxima in a genetic problem are those individuals that get stuck with a pretty good, but not optimal, fitness measure. Any small mutation gives worse fitness. Fortunately, crossover can help them get out of a local maximum. Also, mutation is a random process, so it is possible that we may have a sudden large mutation to get these individuals out of this situation. (In fact, these individuals never get out. It's their offspring that get out of local maxima).

One significant difference between GAs and hill-climbing is that, it is generally a good idea in GAs to fill the local maxima up with individuals. Overall, GAs have less problems with local maxima than back-propagation neural networks.

How Genetic Algorithms search for a solution / how do they work?

It is a random search that is inspired by the evolution of organisms. This generally involves:

- Parallel hill climbing with information exchange among candidate solutions
- Population of candidate solutions

- Crossover for information exchange

The details of how Genetic Algorithms work are explained below:

1. Initialization

While genetic algorithms are generally stated with an initial population that is *generated randomly*, some research has been conducted into using *special techniques* to produce a higher quality initial population. Such an approach is designed to give the GA a good start and speed up the evolutionary process.

Example: Some authors propose a GA for exam timetabling problem in which the GA works only with feasible solutions, implying that the initial population must also be made up of feasible solution. Then the GA is run to improve the fitness of the initial population.

Example: In a simple exam timetabling problem, we can use a non-binary bit string representation to represent the chromosome because it is easy to understand and represent. We use six positions representing six exams with each position's value as the time slot assigned to the exam. We can generate the population randomly to assign each exam a timeslot.

Day	AM		PM	
	time1	time2	time3	time4
Day1			e1	e3
Day2		e5	e6	e2,e4

If we randomly generate six numbers 3, 8, 4, 8, 6, 7 as six timeslots for e1-e6, then the chromosome is 3 8 4 8 6 7.

If the population size is 5, an initial population can be generated randomly as follows:

index	chromosome	fitness	
1	3 8 4 8 6 7	0.005	
2	7 3 7 6 1 3	0.062	
3	5 3 5 5 5 8	0.006	
4	7 6 7 7 2 2	0.020	
5	1 7 4 5 2 2	0.040	

2. Reproduction

There are two kinds of reproduction: generational reproduction and steady-state reproduction.

Generational Reproduction

In generational reproduction, the whole of a population is potentially replaced at each generation. The most often used procedure is to loop $N/2$ times, where N is the population size,

select two chromosomes each time according to the current selection procedure, producing two children from those two parents, finally producing N new chromosomes.

Steady-state Reproduction

The steady-state method selects two chromosomes according to the current selection procedure, performs crossover on them to obtain one or two children, perhaps applies mutation as well, and installs the result back into that population; the least fit of the population is destroyed.

3. Parent Selection mechanism

The effect of selection is to return a probabilistically selected parent. Although this selection procedure is stochastic, it does not imply GA employ a directionless search. The chance of each parent being selected is in some way related to its fitness.

Fitness-based selection

The standard, original method for parent selection is Roulette Wheel selection or fitness-based selection. In this kind of parent selection, each chromosome has a chance of selection that is directly proportional to its fitness. The effect of this depends on the range of fitness values in the current population.

Example: if fitness range from 5 to 10, then the fittest chromosome is twice as likely to be selected as a parent than the least fit.

If we apply fitness-based selection on the population given in example 3.1, we select the second chromosome 7 3 7 6 1 3 as our first parent and 1 7 4 5 2 2 as our second parent.

Rank-based selection

In the rank-based selection method, selection probabilities are based on a chromosome's relative rank or position in the population, rather than absolute fitness.

Tournament-based selection

The original tournament selection is to choose K parents at random and returns the fittest one of these.

4. Crossover Operator

The crossover operator is the most important in GA. Crossover is a process yielding recombination of bit strings via an exchange of segments between pairs of chromosomes. There are many kinds of crossover.

One-point Crossover

The procedure of one-point crossover is to randomly generate a number (less than or equal to the chromosome length) as the crossover position. Then, keep the bits before the number unchanged and swap the bits after the crossover position between the two parents.

Example: With the two parents selected above, we randomly generate a number 2 as the crossover position:

Parent1: 7 3 7 6 1 3

Parent2: 1 7 4 5 2 2

Then we get two children:

Child 1 : 7 3| 4 5 2 2

Child 2 : 1 7| 7 6 1 3

Two-point Cross Over

The procedure of two-point crossover is similar to that of one-point crossover except that we must select two positions and only the bits between the two positions are swapped. This crossover method can preserve the first and the last parts of a chromosome and just swap the middle part.

Example: With the two parents selected above, we randomly generate two numbers 2 and 4 as the crossover positions:

Parent1: 7 3 7 6 1 3

Parent2: 1 7 4 5 2 2

Then we get two children:

Child 1 : 7 3| 4 5| 1 3

Child 2 : 1 7| 7 6| 2 2

Uniform Crossover

The procedure of uniform crossover: each gene of the first parent has a 0.5 probability of swapping with the corresponding gene of the second parent.

Example: For each position, we randomly generate a number between 0 and 1, for example, 0.2, 0.7, 0.9, 0.4, 0.6, 0.1. If the number generated for a given position is less than 0.5, then child1 gets the gene from parent1, and child2 gets the gene from parent2. Otherwise, vice versa.

Parent1: 7 *3 *7 6 *1 3

Parent2: 1 *7 *4 5 *2 2

Then we get two children:

Child 1 : 7 7* 4* 6 2* 3

Child 2 : 1 3* 7* 5 1* 2

5. Inversion

Inversion operates as a kind of reordering technique. It operates on a single chromosome and inverts the order of the elements between two randomly chosen points on the chromosome. While this operator was inspired by a biological process, it requires additional overhead.

Example: Given a chromosome 3 8 4 8 6 7. If we randomly choose two positions 2, 5 and apply the inversion operator, then we get the new string: 3 6 8 4 8 7.

6. Mutation

Mutation has the effect of ensuring that all possible chromosomes are reachable. With crossover and even inversion, the search is constrained to alleles which exist in the initial population. The mutation operator can overcome this by simply randomly selecting any bit position in a string and changing it. This is useful since crossover and inversion may not be able to produce new alleles if they do not appear in the initial generation.

Example: Assume that we have already used crossover to get a new string: 7 3 4 5 1 3. Assume the mutation rate is 0.001 (usually a small value). Next, for the first bit 7, we generate randomly a number between 0 and 1. If the number is less than the mutation rate (0.001), then the first bit 7 needs to mutate. We generate another number between 1 and the maximum value 8, and get a number (for example 2). Now the first bit mutates to 2. We repeat the same procedure for the other bits. In our example, if only the first bit mutates, and the rest of the bits don't mutate, then we will get a new chromosome as below:

2 3 4 5 1 3

Applications of Genetic algorithms

GAs are good across a variety of problem domains. From optimization problems like:

- lens optimization
- prisoner's dilemma
- traveling salesman problem
- automated design - shape optimisation

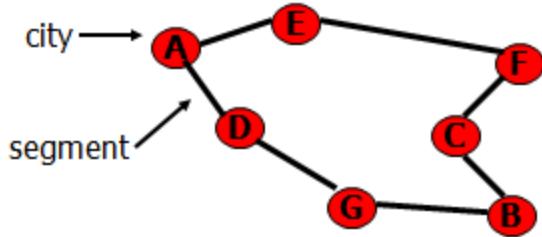
To other like:

- inductive concept learning
- layout planning problems
- routing problems
- control and system identification
- image processing
- marketing, credit and insurance modelling problems, stock prediction, credit scoring, risk assessment etc.

The results can be very good on some problems, and rather poor on others.

Example:

Traveling Salesman Problem



- objective: find the shortest tour that visits each city (NP-hard)
- a tour is encoded by a sequence AEFCBGD which specifies the order in which cities are visited
- fitness: overall length of the tour
- phenotype (tour) is defined by the permutation of genes rather than their values
- usual crossover results in invalid tours

$$\text{AEF|CBGD} \rightarrow \text{AEF DFAB}$$

$$\text{ECG|DFAB} \rightarrow \text{ECG CBGD}$$
- edge recombination operator

$$\text{parent 1 AEFCBDG}$$

$$\text{parent 2 ECGDFAB}$$

City	Connections
A	E,D,F,B
B	C,G,A,E
C	F,B,E,G
D	G,A,F
E	A,F,B,C
F	E,C,D,A
G	B,D,C

1. Select a start city
2. Select the neighbor with the minimum number of connections (break ties randomly)
3. Repeat 2 until no more cities are left

Tutorial 5:

What are the limitations of Genetic algorithms?
Explain constraint handling in genetic algorithms.

PART 6 – KNOWLEDGE BASED SYSTEMS

Njeri Ireri

July – October 2021

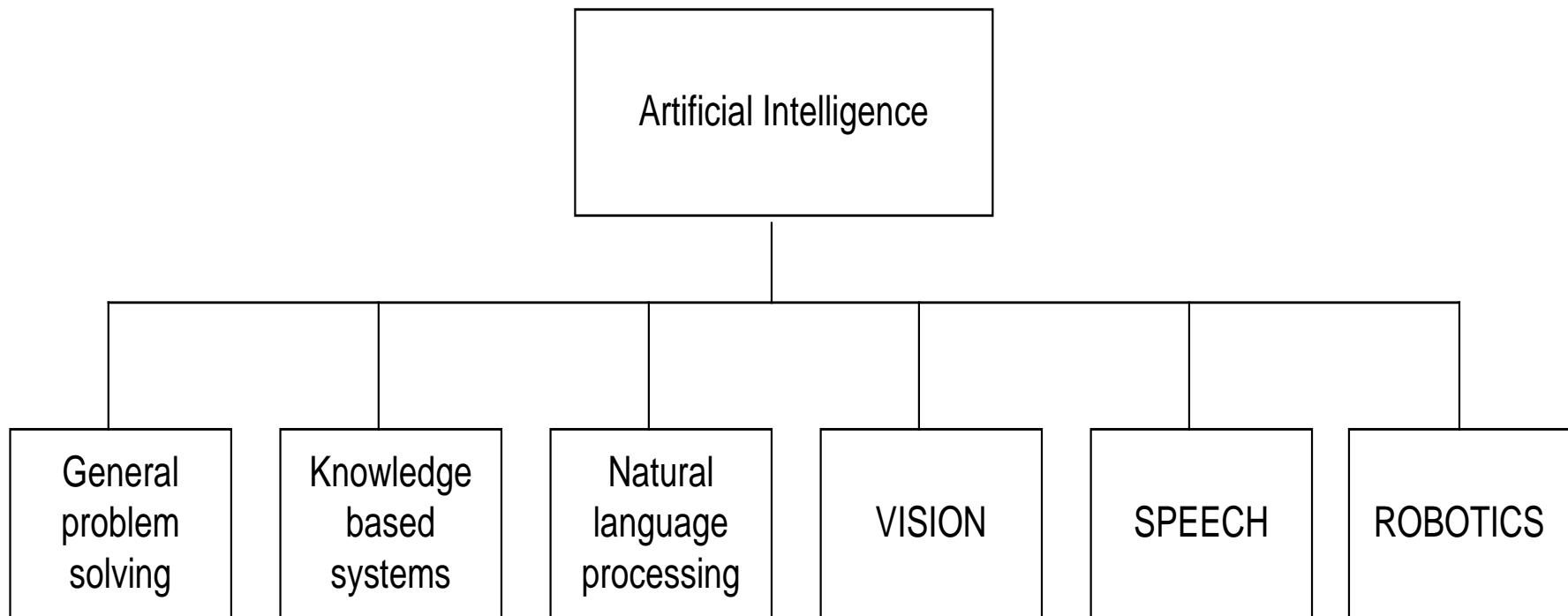
Introduction to KBS

Overview

- AI and KBS
- Definition, Components,
- Application areas

Introduction to KBS

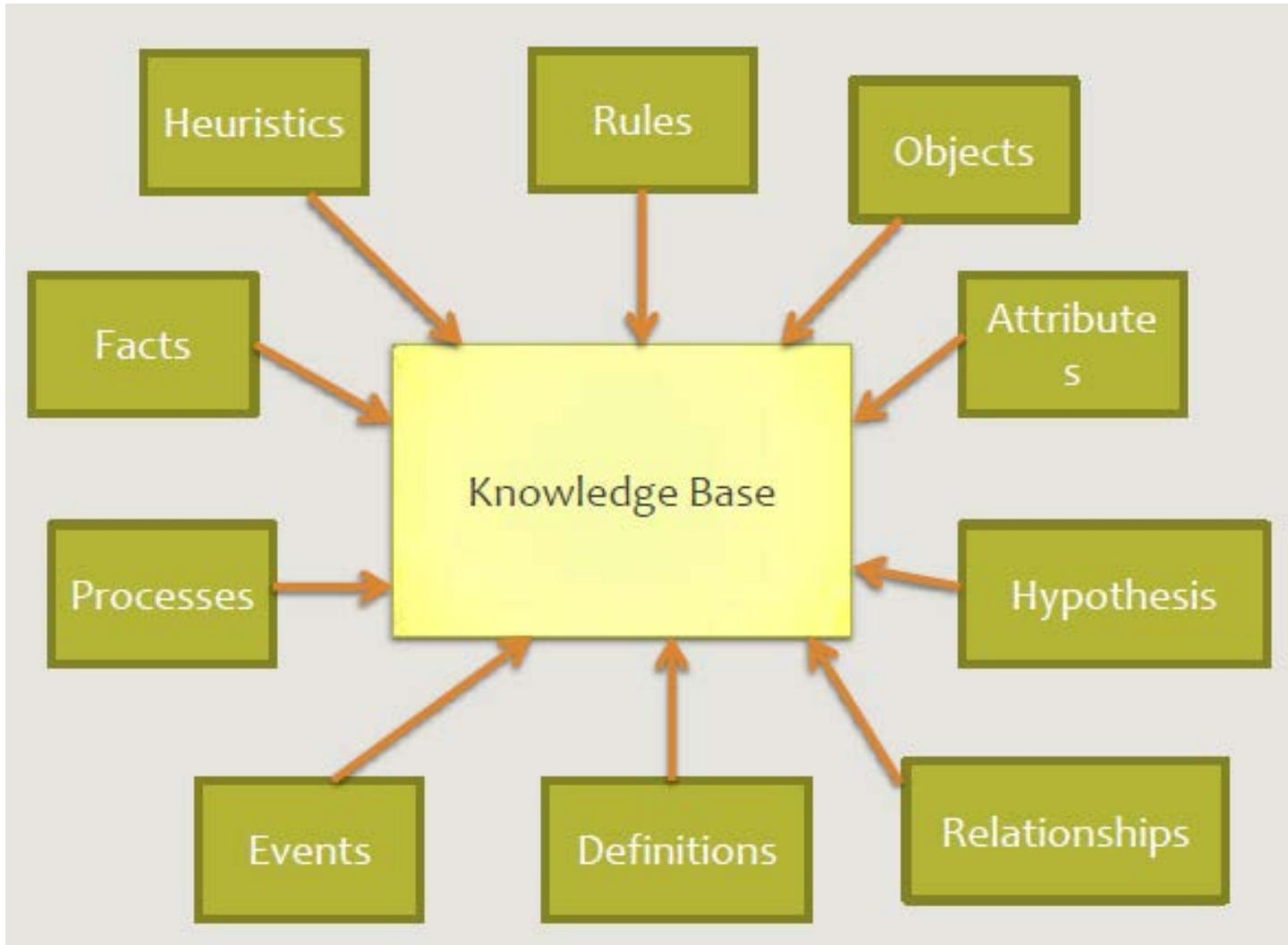
□ Artificial Intelligence and Knowledge Based Systems



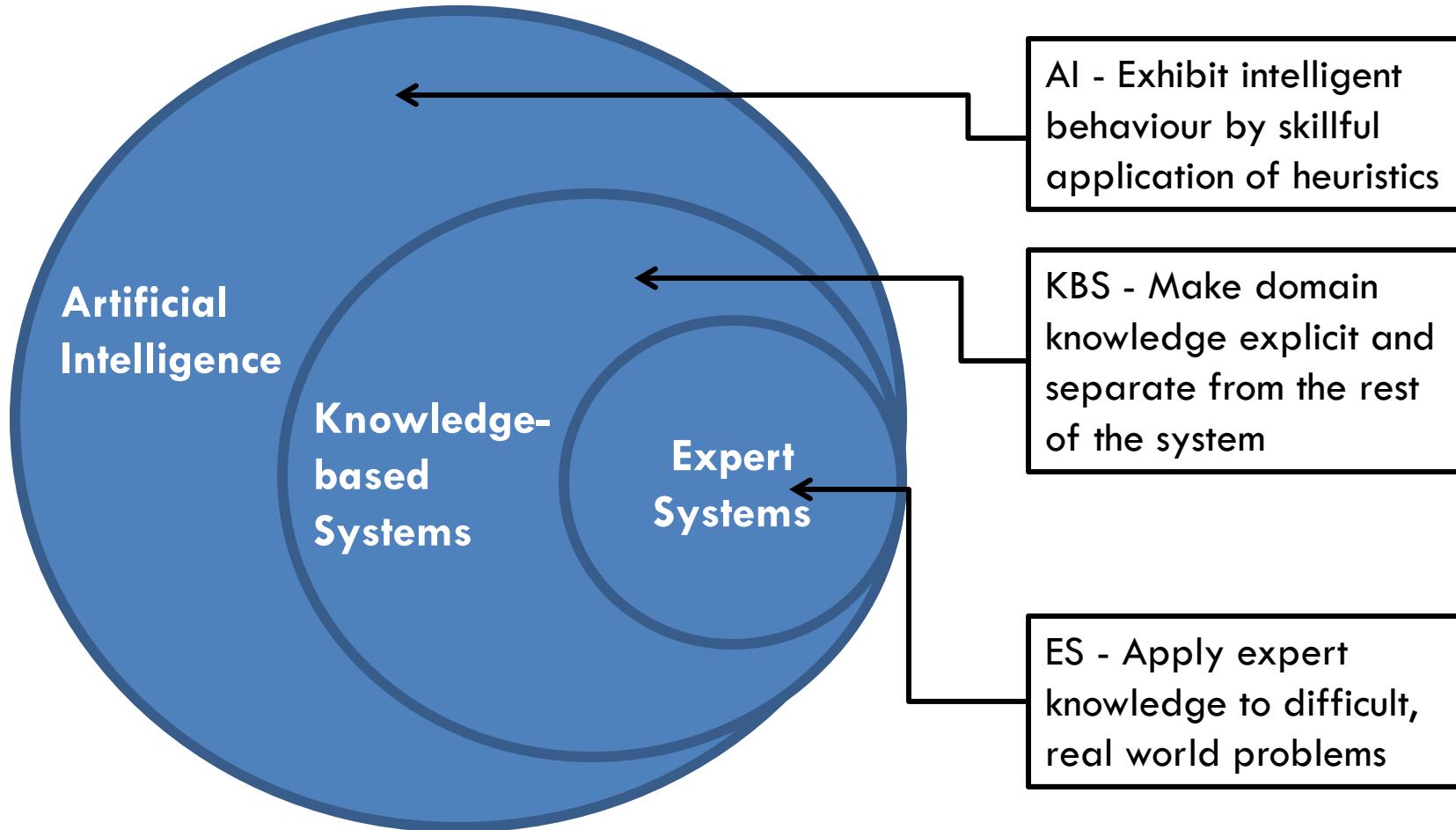
Introduction to KBS

- KBS definition
 - A Knowledge Based Systems is an AI. System that models human expertise in a limited area
 - A system which is built around a knowledge base. i.e. a collection of knowledge, taken from a human, and stored in such a way that the system can *reason* with it
 - A computer system whose usefulness derives primarily from a data base containing *human knowledge in a computerized format*
- Knowledge-based systems are also known as advisory systems, knowledge systems, intelligent job aid systems, or operational systems

Introduction to KBS



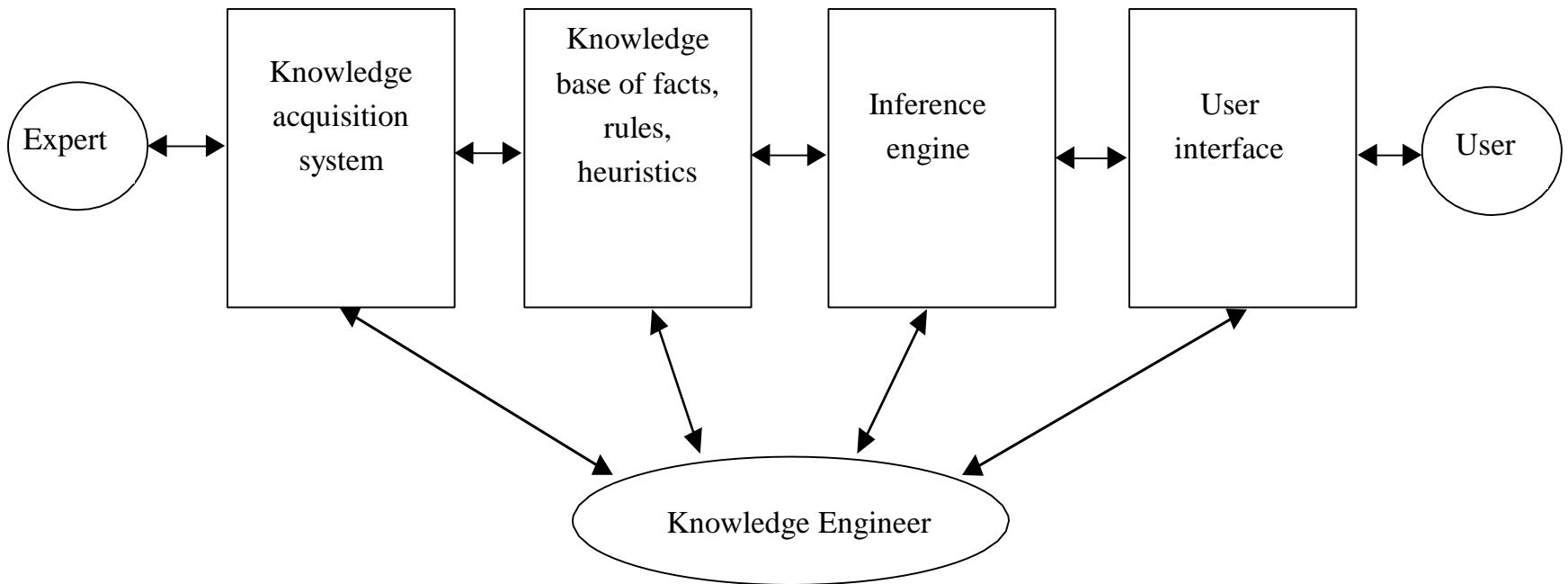
AI, KBS and Expert Systems



Introduction to KBS

- Expert System definition
 - “a computer program that represents and reasons with knowledge of some specialist subject with a view to solving problems or giving advice.” – Peter Jackson
 - “an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solutions.” – Edward Feigenbaum
 - “a system that users human knowledge captured in a computer to solve problems that ordinarily require human expertise.” – Efraim Turban and Jay Aronson

Components and Structure of a KBS



Building expert systems is generally an iterative process - The components and their interaction will be refined over the course of numerous meetings of the knowledge engineer with the experts and users

Creating an KBS

- Two steps involved:
 1. extracting knowledge and methods from the expert (knowledge acquisition)
 2. reforming knowledge/methods into an organised form (knowledge representation)

KBS Application Areas

- Generally, KBS have been found useful for certain sorts of problems. Here are some examples:
 - Interpretation - Inferring situation descriptions from observations OR inferring situation descriptions from sensor data
 - Prediction - Inferring likely consequences of given situations.
 - Diagnosis - Inferring system malfunctions from observations.
 - Design - Configuring objects under constraints.
 - Planning - Developing plans/designing actions.
 - Monitoring - Comparing objects under construction.
 - Debugging - Prescribing remedies for malfunctions.

KBS Application Areas

- Repair - Executing a plan to administer a prescribed remedy.
- Instruction - Diagnosing, debugging, and correcting student performance.
- Advice - providing suitable option(s) under some circumstances
- Control - governing overall system behavior
- Prescription
- Scheduling
- Selection
- Taxonomy

Knowledge Acquisition

We Shall Discuss

- What is Knowledge?
- Sources of Knowledge
- Levels of knowledge
- Categories of knowledge
- What is Knowledge Acquisition?
- Stages of Knowledge Acquisition
- Role of the Knowledge Engineer
- Methods of Knowledge Acquisition
- Knowledge Acquisition Difficulties
- Organizing the Knowledge

Data, Information and Knowledge

- What is knowledge?

- Data:

- Raw facts, figures, measurements

- Information:

- Refinement and use of data to answer specific question

- Knowledge:

- Refined information

- True rational belief(philosophy).OR facts, data and their relationships (Computational view)

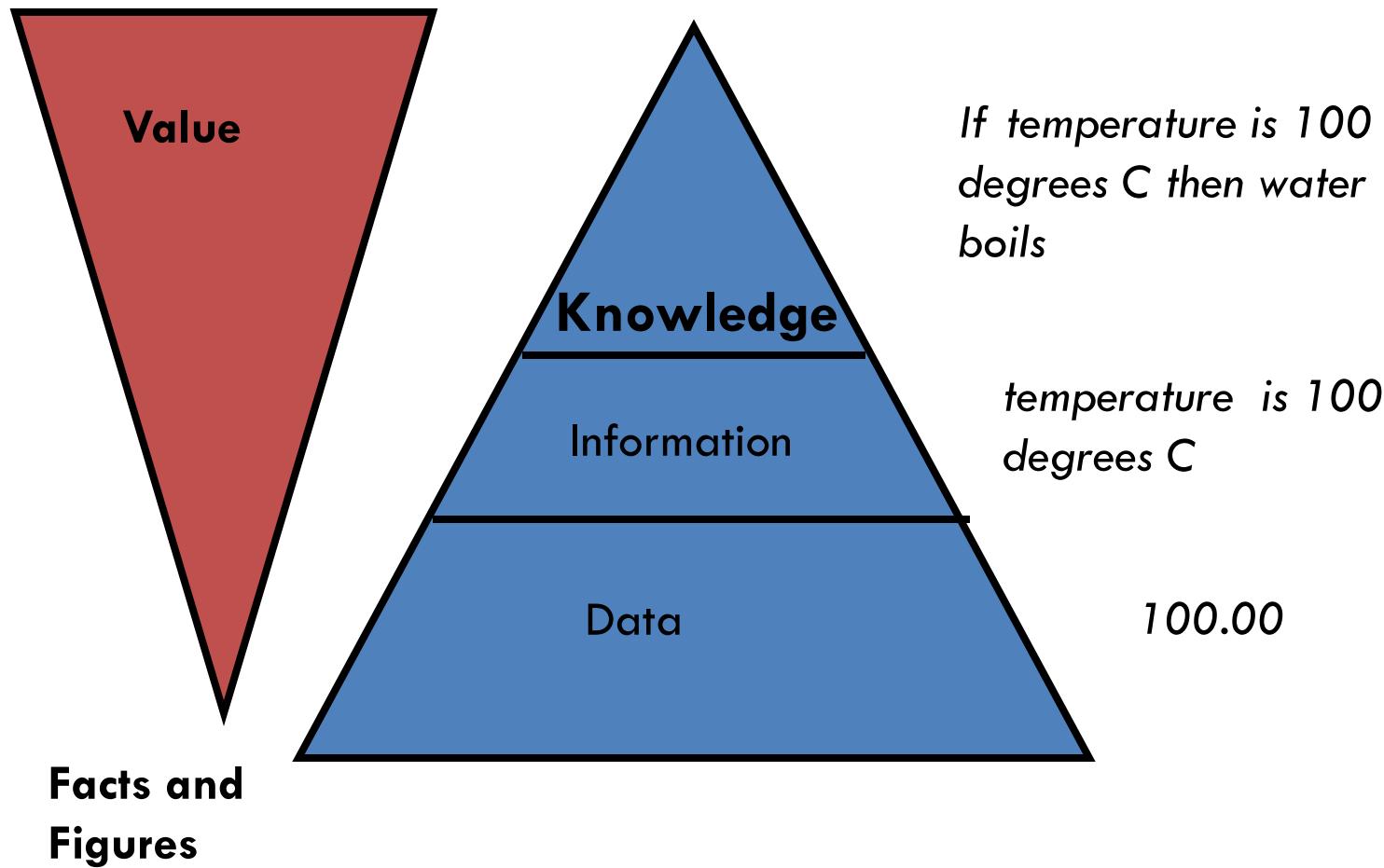
- Knowledge is the sort of information that people use to solve problems.

- Knowledge includes: facts, concepts, procedures, models, heuristics, examples.

Data, Information and Knowledge

Concepts

Example



Sources of Knowledge

- **Documented**
 - books, journals, procedures
 - films, databases
- **Undocumented**
 - people's knowledge and expertise
 - people's minds, other senses
 - This is mainly **tacit/implicit** knowledge (e.g. procedural knowledge) that is difficult to capture
 - It is implied by or inferred from actions or statements of experts

Knowledge Acquisition

- **Knowledge acquisition** is the process of extracting, structuring and organizing **knowledge** from one source, usually human experts, so it can be used in Knowledge based systems
- This is often the major obstacle in building a KBS Systems
 - Knowledge acquisition is a multifaceted problem that encompasses many of the technical problems of knowledge engineering, the enterprise of building knowledge base systems (Gruber)

Knowledge Acquisition

- The knowledge acquisition process is usually comprised of three principal stages:
 - ▣ **Knowledge elicitation** is the interaction between the expert and the knowledge engineer/program to elicit the expert knowledge in some systematic way
 - ▣ The knowledge thus obtained is usually stored in some form of human friendly **intermediate representation**
 - ▣ The intermediate representation of the knowledge is then compiled into an **executable form** (e.g. production rules) that the inference engine can process

Context - Developing Expert Systems

- Who is involved?

- Knowledge Engineer

- A knowledge engineer is a computer scientist who knows how to design and implement programs that incorporate artificial intelligence techniques
 - Interacts between expert and Knowledge Base
 - Needs to be skilled in extracting knowledge
 - Uses a variety of techniques

- Domain Expert

- A domain expert is an individual who has significant expertise in the domain of the expert system being developed

Methods of Knowledge Acquisition

- Manual:
 - interview with experts
 - structured, semi structured, unstructured interviews
 - track reasoning process and observing
- Semi Automatic:
 - use a computerised system to support and help experts and knowledge engineers
 - Repertory Grid Analysis
- Automatic:
 - minimise the need for a knowledge engineer or expert
 - Broadly, this means **using a computer program to convert data into knowledge**. This process may also be described as **learning**
 - Induction Decision Trees (ID3)

Knowledge Acquisition Difficulties

- Transferring knowledge from one person to another is difficult. Even more difficult in A.I For these reasons:
 - expressing knowledge
 - The problems associated with transferring the knowledge to the form required by the knowledge base
 - experts busy or unwilling to part with knowledge
 - methods for eliciting knowledge not refined
 - collection should involve several sources not just one
 - it is often difficult to recognise the relevant parts of the expert's knowledge
 - experts change

Knowledge Representation

Overview

- Definition
- Representation schemes/formalisms

Definition

- **Knowledge.** True rational belief(philosophy). *OR* facts, data and relationships (Computational view).
- **Representation.** Structure + operations; *OR* map + operations; *OR* game layout and rules of play; *OR* abstract data types.
- **Knowledge representation.** Framework for storing knowledge and manipulating knowledge *OR* ‘Set of syntactic and semantic conventions that makes it possible to describe things.’ Bench-Capon, 1990.

Knowledge-based Agent

- Central component of a knowledge-based agent is its **knowledge-base (KB)**
- A KB is a **set of representations of facts about the world**
- Each individual representation is called a **sentence**
- The sentences are expressed in a language called a **knowledge representation language**
- A knowledge-based agent should be able to **infer**

Knowledge Representation

- The object of Knowledge Representation is to **express knowledge in a computer-tractable form**, so that it can be used to help agents perform well.
- A Knowledge Representation language is defined by two aspects:
 - **Syntax:** describes how to make sentences OR describes the possible configurations that can constitute sentences.
 - **Semantics:** determine the facts in the world to which the sentences refer OR the “things” in the sentence.

Knowledge Representation Schemes

- **Different Knowledge Representation schemes/formalisms**
 - Natural Language
 - Rules
 - Logic
 - Propositional logic (Boolean Logic)
 - Predicate logic (First Order Logic)
 - Semantic Nets
 - Frames

Natural Language

Expressiveness of natural language:

- Very expressive, probably everything that can be expressed symbolically can be expressed in natural language (pictures, content of art, emotions are often hard to express)
- Probably the most expressive knowledge representation formalism we have. Reasoning is very complex, hard to model

Problems with natural language:

- Natural language is often ambiguous
- The syntax and semantics are not fully understood
- There is little uniformity in the structure of sentences

Natural Language is Ambiguous

Examples:

- Mary lost a jewel.
 - *Jewel is noun or a verb?*
- Lung cancer in women mushrooms
 - *Mushrooms is noun or a verb?*
- Iraqi Head Seeks Arms
 - *Arms can mean different things, which is it?*
- Two Soviet Ships Collide, One Dies
 - *What does one refer to in this case?*
- Chef throws his heart into feeding needy
 - *Throws his heart is not decomposed normally in this case: idiom*

Rules

- These are formalizations often used to specify recommendations, give directives or strategy.
 - **Format:** **IF <premises> THEN <conclusion>.**
 - Related ideas: rules and fact base; conflict set - source of rules; conflict resolution- deciding on rules to apply.
-
- Advantages: easy to use; explanations are possible; capture heuristics; can handle uncertainties to some extent.
 - Disadvantages: cannot cope with complex associated knowledge; they can grow to unmanageable size.

Rules

Consists of:

- a rule set for representing the expert knowledge
- a “database management system” for the case-specific facts
- a rule interpreter for problem solving

Example

IF: (1) stain of organism is Gram negative AND

(2) morphology of organism is rod AND

(3) aerobicity of organism is aerobic

THEN: strong evidence (0.8) that organism is Enterobact

Properties of rule-based systems:

- modularity of rule, very expressive, easy handling of certainty factors (probabilistic, possibilistic reasoning)
- Lack of precise semantics of rules. Not always efficient

Rules - A simple example

- Assume: Knowledge base consisting of facts and rules, a rule interpreter to match the rule conditions against facts and means for executing the rules.

Rules:

R1: IF: Raining,Outside(x),Has_Umbrella(x)

THEN: Uses_Umbrella(x)

R2: IF: Raining,Outside(x)

NOT Has_Umbrella(x)

THEN: Wet(x)

R3: IF: Wet(x)

THEN: Gets_Cold(x)

R4: IF: Sunny,Outside(x)

THEN: Gets_Sun_Tan(x)

Initial facts: Raining, Outside(John)

Rules - A simple example

Correct:

- Only one rule, R2 matches the facts with $[x \rightarrow \text{John}]$, hence add **Wet(John)**
- Facts after first cycle:
Raining, Outside(John), Wet(John)
- Now R3 matches facts, hence add
Gets_Cold(John)
- Facts after second cycle:
Raining, Outside(John), Wet(John), Gets_Cold(John)

Incorrect:

Gets_Sun_Tan(John)

Process of deriving new facts from given facts, is called **INFERENCE**

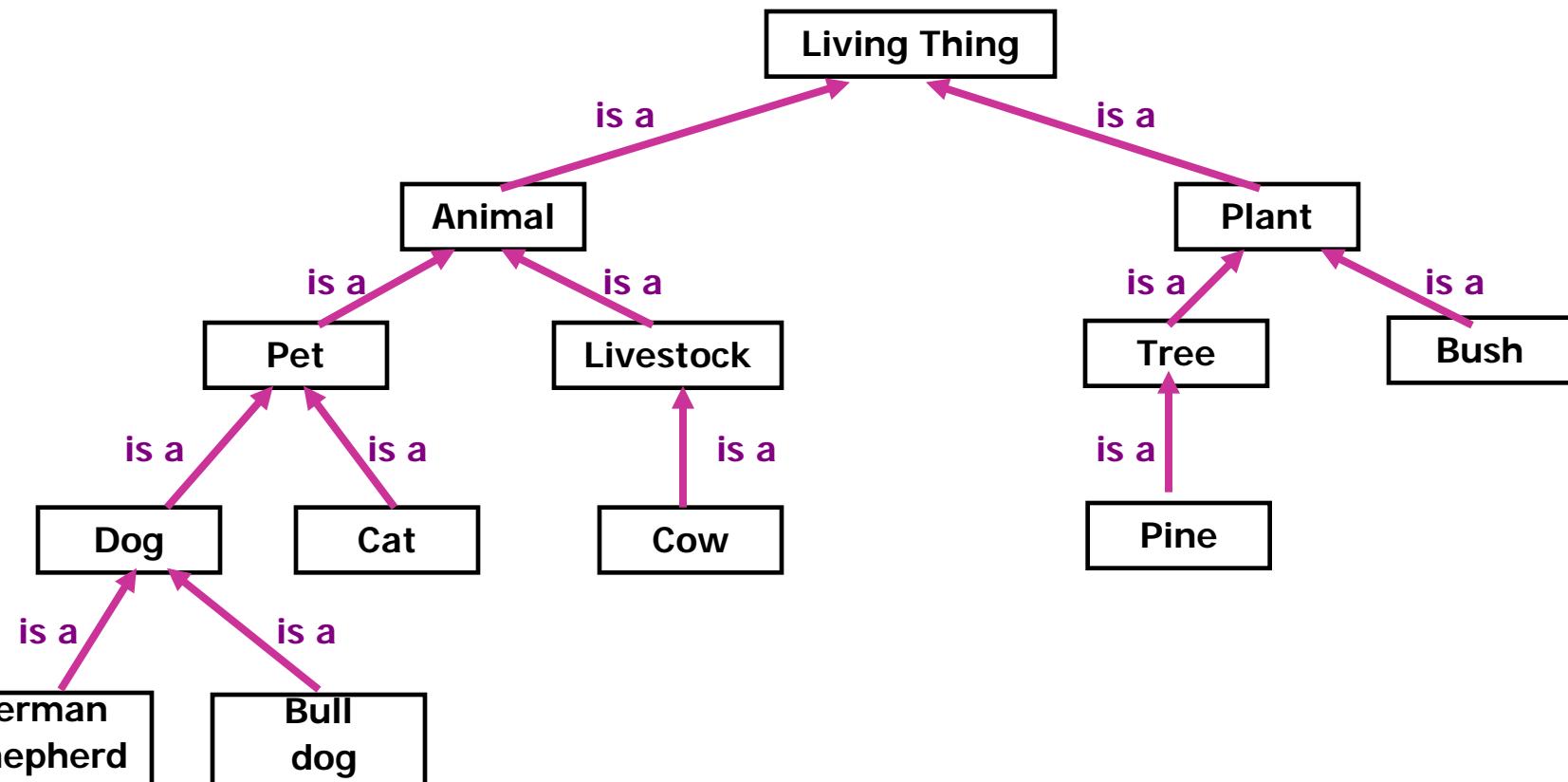
Semantic Networks

- Semantic networks are graphical representation of entities and their relationships. The nodes are objects or events; the arcs are the relationships or moves.

- Advantages. Easy to translate to predicate calculus.
- Disadvantages. Cannot handle quantifiers; nodes may have confusing roles or meanings; searching may lead to combinatorial explosion; cannot express standard logical connectives; can represent only binary or unary predicates.

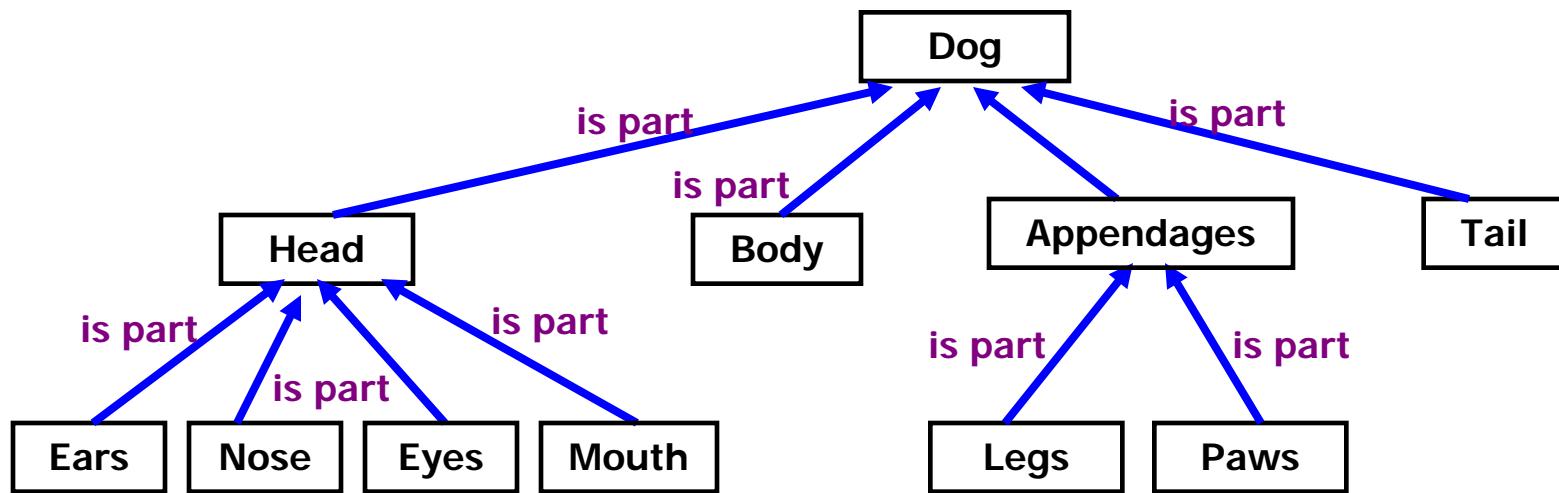
Semantic Networks - An example 1

IS - A Hierarchy

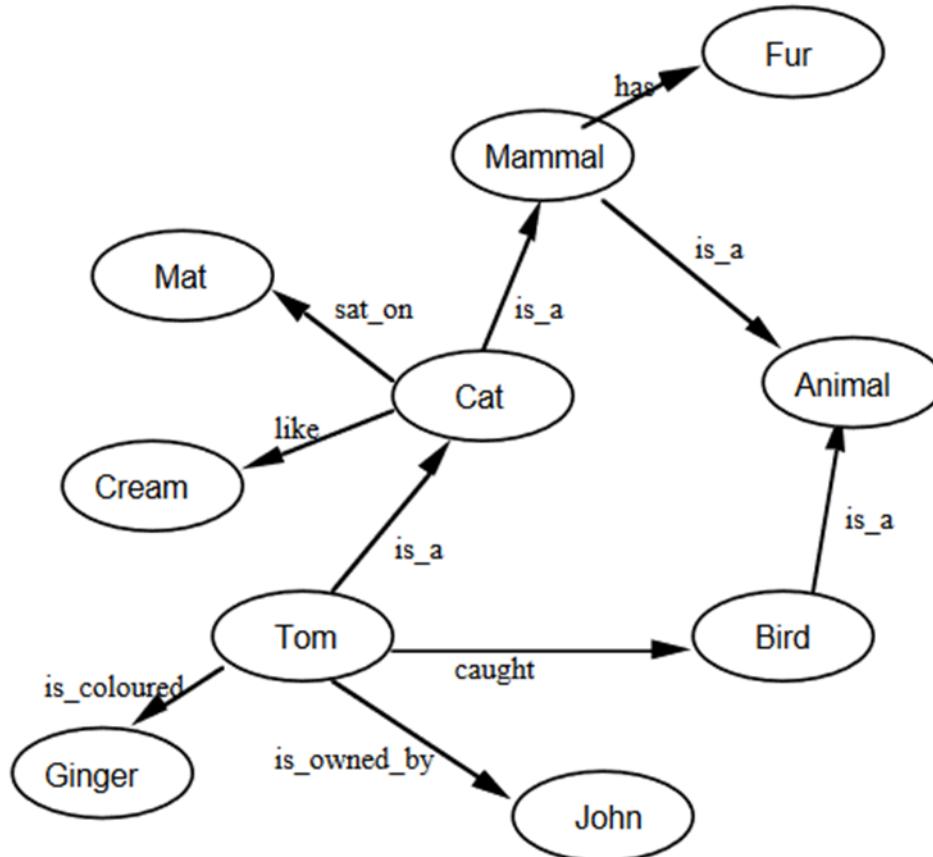


Semantic Networks - An example 2

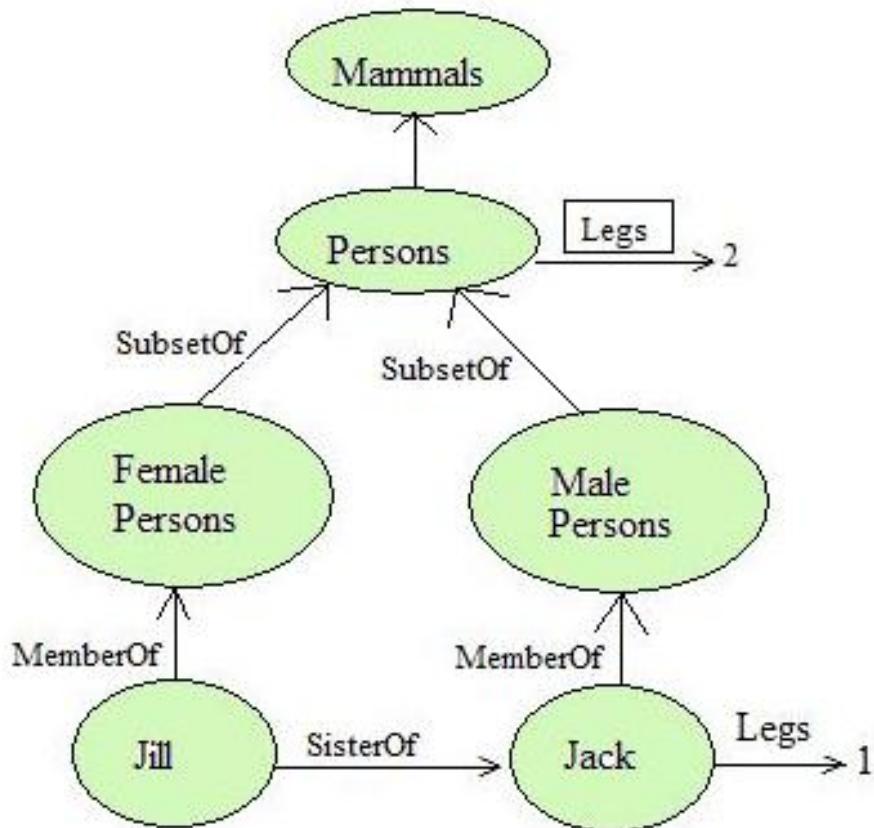
IS - PART Hierarchy



Semantic Networks - An example 3



Semantic Networks - An example 4



Propositional Logic

- **Propositional Logic (Propositional calculus/Boolean Logic)** - assertions describing things, use logical connectives and boolean logic
 - Advantages. Can reason about the world; based on proven theory
 - Disadvantages. Components cannot be individually examined

Propositional Logic

The syntax:

- Vocabulary:
 - A set of propositional symbols - e.g., P, Q, ...
- A set of logical connectives or operators
 - usually \vee (OR), \wedge (AND), \neg (NOT), \rightarrow (implication), maybe \Leftrightarrow (equivalence), Parenthesis (for grouping)

Read: Logic	\vee	\wedge	\rightarrow	\neg
Nat. Lang.	Or	And	Implies	Not

- The special symbols
 - True, False (logical constants)

Propositional Logic

Rules for forming sentences:

- Each symbol (i.e., a constant or a propositional symbol) is a sentence (an atomic sentence).
- A sentence in parentheses is a sentence.
- If P and Q are sentences, then so are
 - $P \vee Q$ (disjunction)
 - $P \wedge Q$ (conjunction)
 - $\neg P$ (negation)
 - $P \rightarrow Q$ (implication)
 - and similarly for whatever other connectives we allow

Propositional Logic

Sample sentences

- P
- True
- $P \vee Q$
- $\neg P$
- $(P \vee Q)$
- $\neg(P \vee Q)$
- $\neg P \vee Q$
- $(P \vee Q) \rightarrow R$
- $P \wedge \neg P$

What do the sentences mean?

- E.g., P might be “It is raining in Nyahururu”, and Q, “Nairobi is a city”
- We interpret logical connectives in the obvious way.
 - E.g., $\neg P$ means that P is not the case; $P \vee Q$ means that at least one of P or Q is true.

Propositional Logic - example

- Let, Fact P: "Ali likes chips"
- Let, Fact Q: "Ali eats chips"

- Other possible facts:
 - $P \vee Q$: "Ali likes chips or Ali eats chips"
 - $P \wedge Q$: "Ali likes chips and Ali eats chips"
 - $\neg Q$: "Ali doesn't eat chips"
 - $P \rightarrow Q$: "If Ali likes chips then Ali eats chips"

Propositional Logic

Truth

- For sentences, we also get to say whether they are true or false.
 - ▣ True is always true; False always false.
 - ▣ P, Q, etc., are true or false depending on their interpretation.
 - So these are satisfiable, but not valid.
 - ▣ Complex sentences are true or false as a function of their connective.
 - Usually specified as a truth table.

Propositional Logic

Truth tables

Conjunction(\wedge)

P	Q	$P \wedge Q$
false	false	false
false	true	false
true	false	false
true	true	true

Disjunction(\vee)

P	Q	$P \vee Q$
false	false	false
false	true	true
true	false	true
true	true	true

Implication(\rightarrow)

P	Q	$P \rightarrow Q$
false	false	true
false	true	true
true	false	false
true	true	true

Negation(\neg)

P	$\neg P$
true	false
false	true

Equivalence(\leftrightarrow)

P	Q	$P \leftrightarrow Q$
false	false	true
false	true	false
true	false	false
true	true	true

Propositional Logic

A Proof Theory for Propositional Logic

- It is easy to devise a procedure to determine the truth of an arbitrary sentence in propositional logic.
- Just write down a big truth table, and see if the sentence is always true.
- E.g., suppose we want to know if $\neg(P \wedge Q) \rightarrow \neg P \vee \neg Q$.
- Here is a truth table:

P	Q	$\neg(P \wedge Q)$	$\neg P \vee \neg Q$	$\neg(P \wedge Q) \rightarrow \neg P \vee \neg Q$
false	false	true	true	true
false	true	true	true	true
true	false	true	true	true
true	true	false	false	true

So we have proved this sentence.

Propositional Logic

Reasoning in Propositional Logic

- Similarly, if we assume a few things, we can determine if something follows.
- E.g., if we assume P , then $P \vee Q$, say, degenerates into $\text{True} \vee Q$, which a truth table will tell us is always true.
- So, we can always draw valid conclusions from premises, **regardless of what any of this means**

Propositional Logic

Reading Assignment - Common Inference Rules

- Modus Ponens: mode that affirms

$$\alpha \rightarrow \beta, \alpha \vdash \beta$$

- And-Elimination: simplification

$$\alpha_1 \wedge \alpha_2 \vdash \alpha_i$$

- And-Introduction: conjunction

$$\alpha_1, \alpha_2 \vdash \alpha_1 \wedge \alpha_2$$

- Or-Introduction: addition

$$\alpha \vdash \alpha \vee \beta$$

- Double-Negation Elimination:

$$\neg\neg \alpha \vdash \alpha$$

- Resolution:

$$\alpha \vee \beta, \neg \beta \vee \gamma \vdash \alpha \vee \gamma$$

Propositional Logic

- Propositional Logic has limitations, - it is not expressive enough
- First-Order Logic is an improvement and is useful

Predicate Logic (FOL)

- **Predicate Logic (Predicate Calculus / First Order Logic)** - is an extension of propositional Logic.
- **Predicates** that are used in FOL are of the form function(arguments), where function is any object or relationship and quantifiers are used.
- In FOL, the world consists of **objects**, i.e. things with individual identities and **properties** that distinguish them from other objects.
- Among these objects, various **relations** hold. Some of these relations are **functions**. i.e. relations in which there is only one “value” for a given “input”.

First-Order Logic (FOL)

Examples:

- Objects: people, houses, colours, the moon, Mutua,...
 - Relation: brother of, bigger than, inside, part of, owns,..
 - Properties: red, round, healthy, tall..
 - Functions: father of, best friend,...
-
- Advantages. It has well defined rules for manipulation; it is expressive.
 - Disadvantages. Cannot handle uncertainty; uses small primitives for descriptions whose numbers can be many.

First-Order Logic

Read: Logic	\vee	\wedge	\rightarrow	\neg	\forall	\exists
Nat. Lang.	Or	And	Implies	Not	Forall	Exists

- Examples:
- (a) man(Pat)
 - (d) man(Jan) \vee woman(Jan)
 - (b) married(Pat,Jan)
 - (e) $\forall x \exists y [\text{person}(x) \rightarrow \text{has_mother}(x,y)]$
 - (c) $\forall x \forall y [[\text{married}(x,y) \wedge \text{man}(x)] \rightarrow \neg \text{man}(y)]$

Properties of First-Order Logic:

- very expressive as well as unambiguous syntax and semantics
- no generally efficient procedure for processing knowledge

Correct:

raining
raining \rightarrow street_wet
street_wet

Incorrect:

raining
raining \rightarrow street_wet
elephant_hungry

First-Order Logic

Syntax:

- Connectives - $\Rightarrow, \wedge, \vee, \Leftrightarrow$
- Quantifiers - \forall, \exists
- Constants - A, X₁, Mutua, Alice,..
- Variables - a, x, s,..
- Predicate - Before, HasColour, Raining,..
- Functions - Mother, LegOf,..

First-Order Logic

Connections between \forall and \exists

- e.g. “Everybody likes ice cream” means that there is no one who doesn’t like ice cream.
 - ▣ $\forall x \text{ likes}(x, \text{icecream}) \Leftrightarrow \neg \exists x \neg \text{likes}(x, \text{icecream})$
- Using De Morgan’s Theorem:
 - $\forall x S \Leftrightarrow \neg \exists x \neg S$
 - $\exists x S \Leftrightarrow \neg \forall x \neg S$

Examples: Translate into Logic

- Some dogs bark

$$\exists x. \text{Dog } x \wedge \text{Barks } x$$
$$\exists xy . \text{Dog } x \wedge \text{Bark } y \wedge \text{makes_sound } x$$

- All dogs have four legs

$$\begin{aligned} \forall x . \text{Dog } x \rightarrow \exists y z w u . & \text{Leg } y \wedge \text{Leg } z \wedge \text{Leg } w \wedge \text{Leg } u \\ & \wedge \text{has } x y \wedge \text{has } x z \wedge \text{has } x w \wedge \text{has } x u \wedge y \neq z \wedge y \neq \\ & w \wedge y \neq u \wedge z \neq w \wedge z \neq u \wedge w \neq u \end{aligned}$$

- All barking dogs are irritating.

$$\forall x . \text{Dog } x \wedge \text{Barking } x \rightarrow \text{Irritating } x$$

Examples: Translate into Logic

- No dogs purr.
$$\neg \exists x . \text{Dog } x \wedge \text{Purrs } x$$
$$\forall x . \text{Dog } x \rightarrow \neg \text{Purrs } x$$
- Fathers are male parents with children
$$\forall x . \text{Male } x \wedge \text{Parent } x \wedge (\exists y . \text{Child } y \text{ and has } x y) \leftrightarrow \text{Father } x$$
- Students are people who are enrolled in courses.
$$\forall x . \text{Student } x \leftrightarrow \text{Person } x \wedge (\exists y . \text{Course } y \wedge \text{enrolled_on } x y)$$

Frames

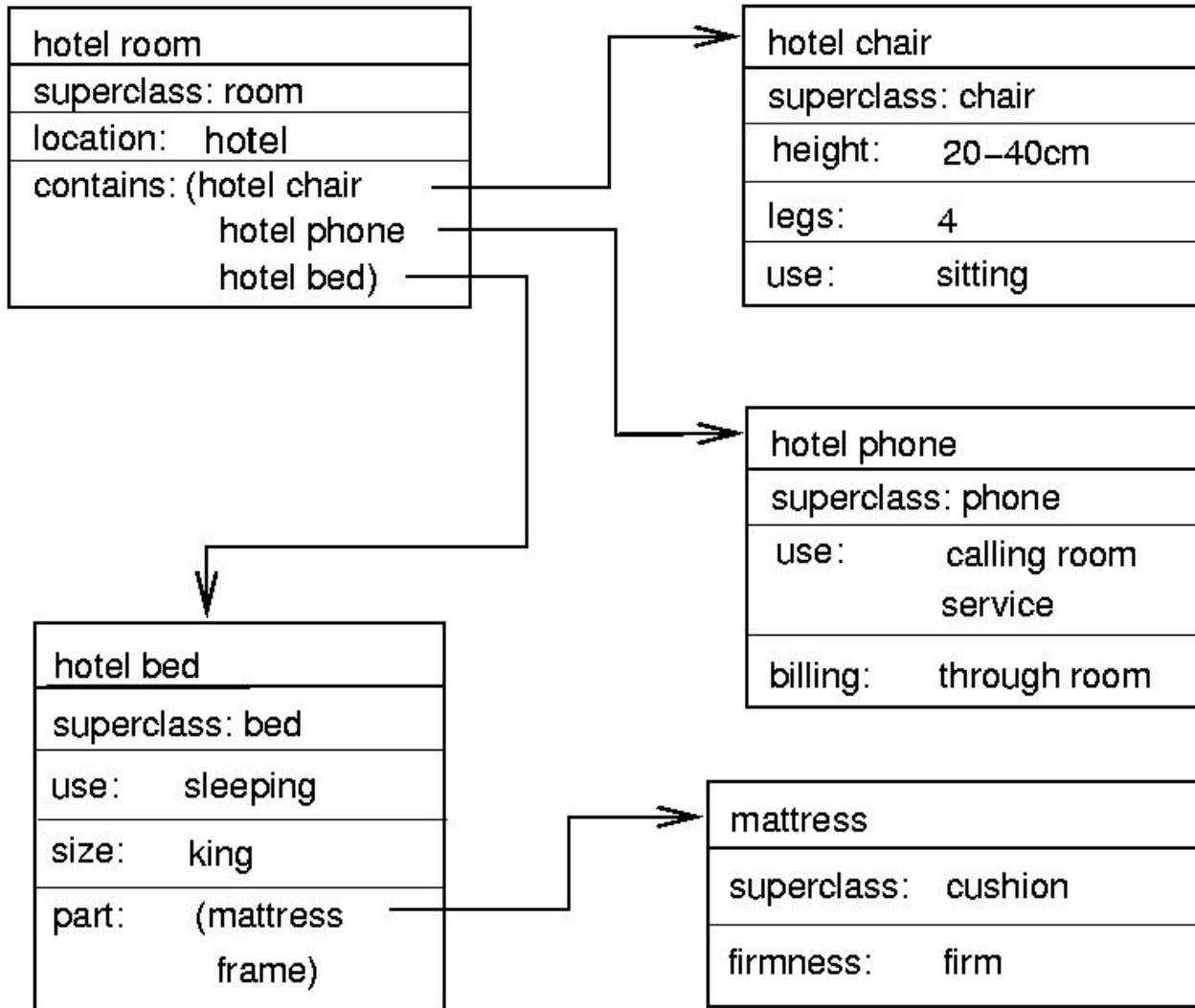
- A frame is an AI data structure used to divide knowledge into substructures which represent “stereotyped situations.”
- Frames are also useful for representing commonsense knowledge
- A frame represents an object that is typical to a stereotypical situation
- Objects are arranged in a hierarchical manner
- Frames can be derived from semantic nets
- A frame comprises of slots and slot filler

Frames - example 1

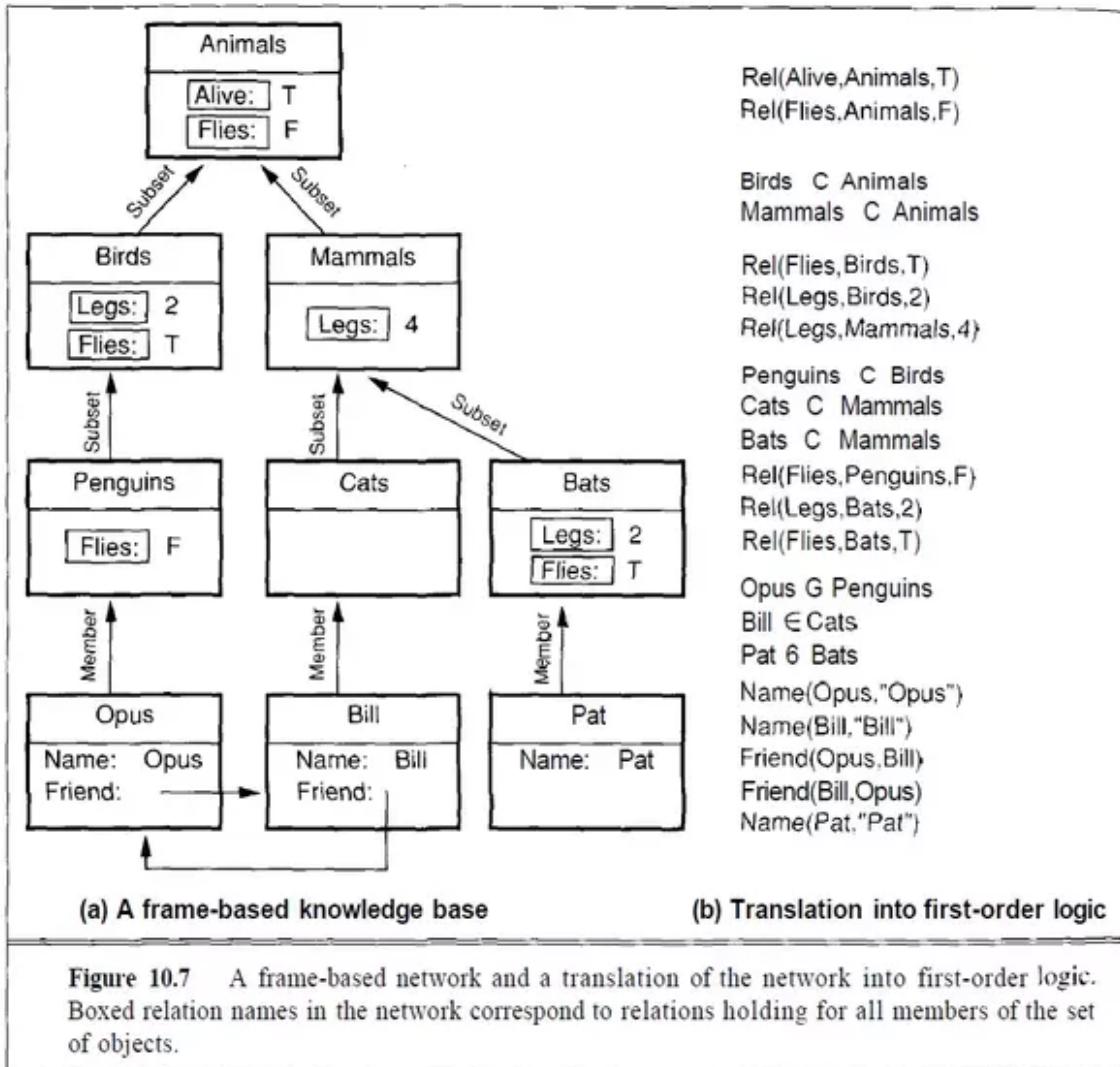
Slots	Fillers
publisher	Thomson
title	Expert Systems
author	Giarratano
edition	Third
year	1998
pages	600

Frame Name	Vacation
Where	Albury
When	March
Cost	\$1000

Frames - example 2



Frames - example 3



Representation of Knowledge -

Summary

- There is **no single most adequate** knowledge representation formalism/scheme for everything.
- Main points for selecting a representation formalism: **what** should be represented, **how** should the knowledge be processed.
- There are many more representation formalisms. All the above mentioned are **symbolic**. There are non-symbolic (e.g. pictorial) ones. Neural networks work on non-symbolic representations.

ARTIFICIAL INTELLIGENCE

PART 7 – LEARNING

Njeri Ireri

July – October 2021

Overview

- Introduction to machine learning
- Machine Learning Models
- Machine Learning Methods
- Machine Learning Paradigms

Learning & Adaptation

- Learning in general:
"Modification of a behavioral tendency by expertise."
(Webster 1984)
- Learning Machine:
"A learning machine, broadly defined is any device whose actions are influenced by past experiences." (Nilsson 1965)
- Machine Learning: (how does a machine learn – slide 3)
"Any change in a system that allows it to perform better the second time on repetition of the same task or on another task drawn from the same population." (Simon 1983)
"An improvement in information processing ability that results from information processing activity." (Tanimoto 1990)

Machine Learning

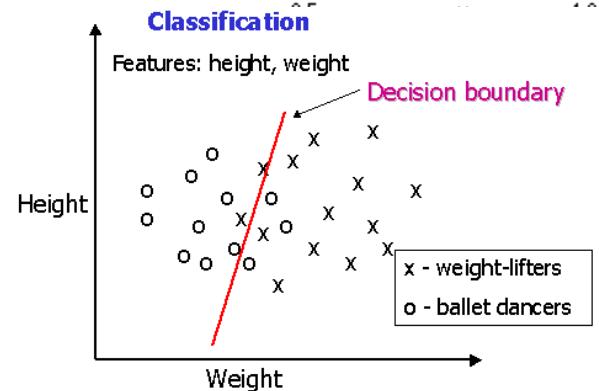
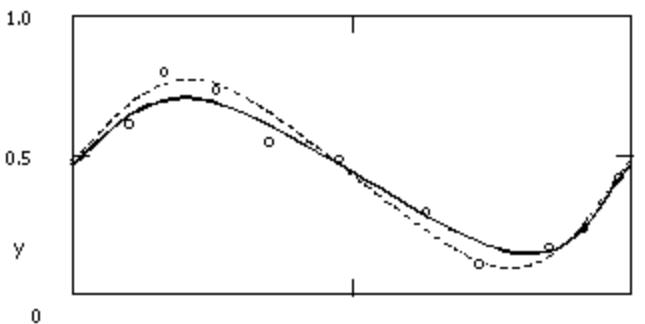
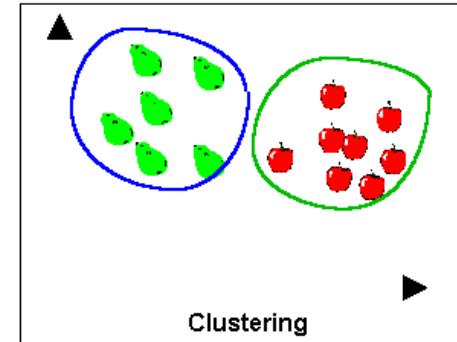
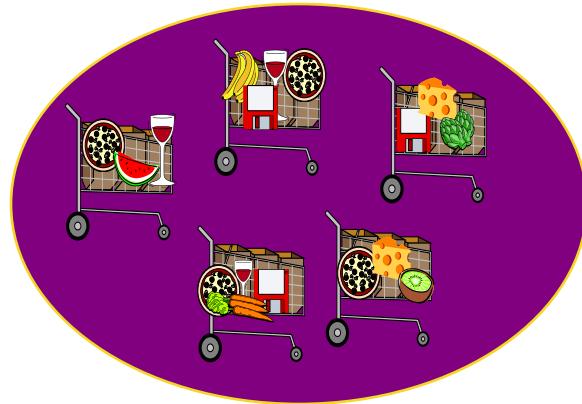
- Machine learning involves automatic procedures that learn a task from a series of examples
- Most convenient source of examples is **data**

Definition:

- A computer program is said to **learn** from **experience E** with respect to some class of **tasks T** and **performance measure P**, if its performance at tasks in T, as measured by P, improves with experience.

Machine Learning Models

- Classification
- Regression
- Clustering
- Time series analysis
- Association Analysis
- Sequence Discovery
-

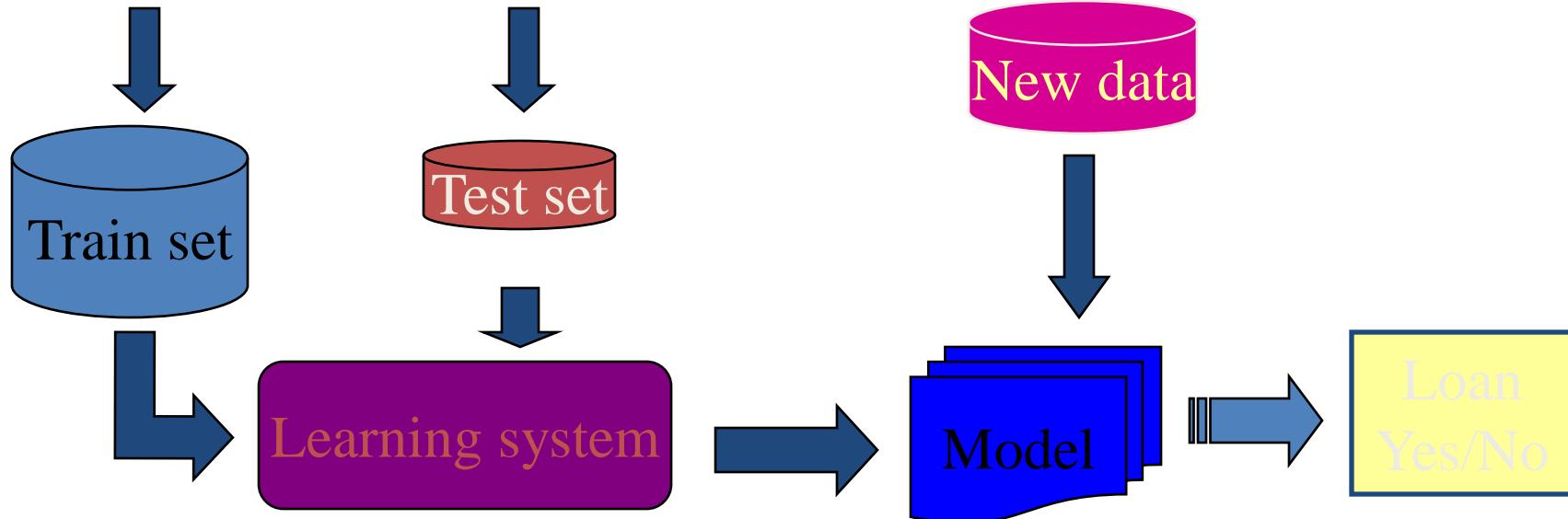


Classification example

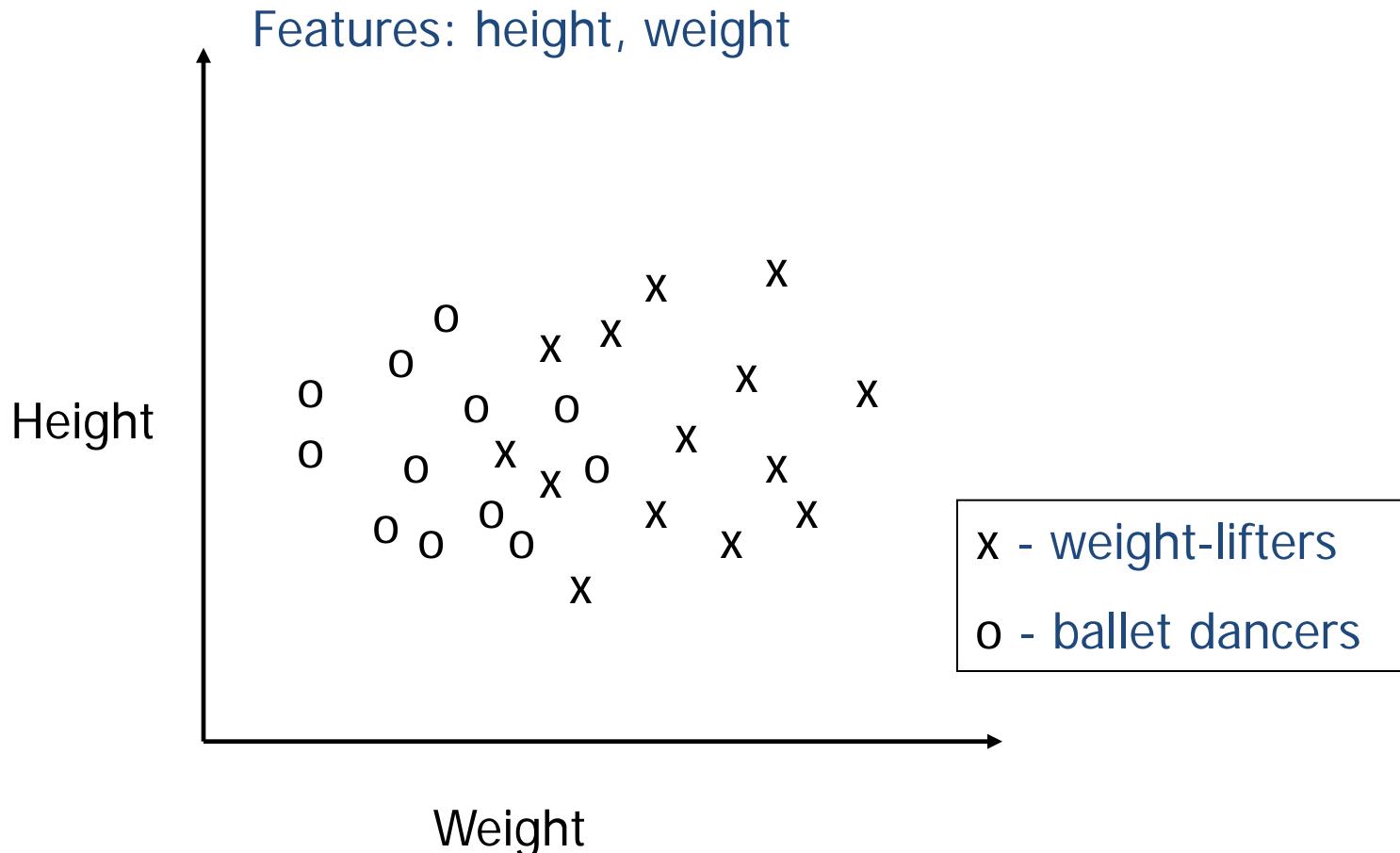
Loan Application Appraisal

No	Sex	Age	Marital status	Net Income	...	Loan
1	F	38	Married	45K	.	Yes
2	M	42	Married	66K	.	Yes
3	F	52	Single	43K	.	No
4	M	50	Single	70K	.	No
5	F	27	Married	40K	.	No
6	M	45	Divorced	38K	.	No
7	F	35	Widow	59K	.	Yes
8	M	32	Married	52K	.	Yes
..						

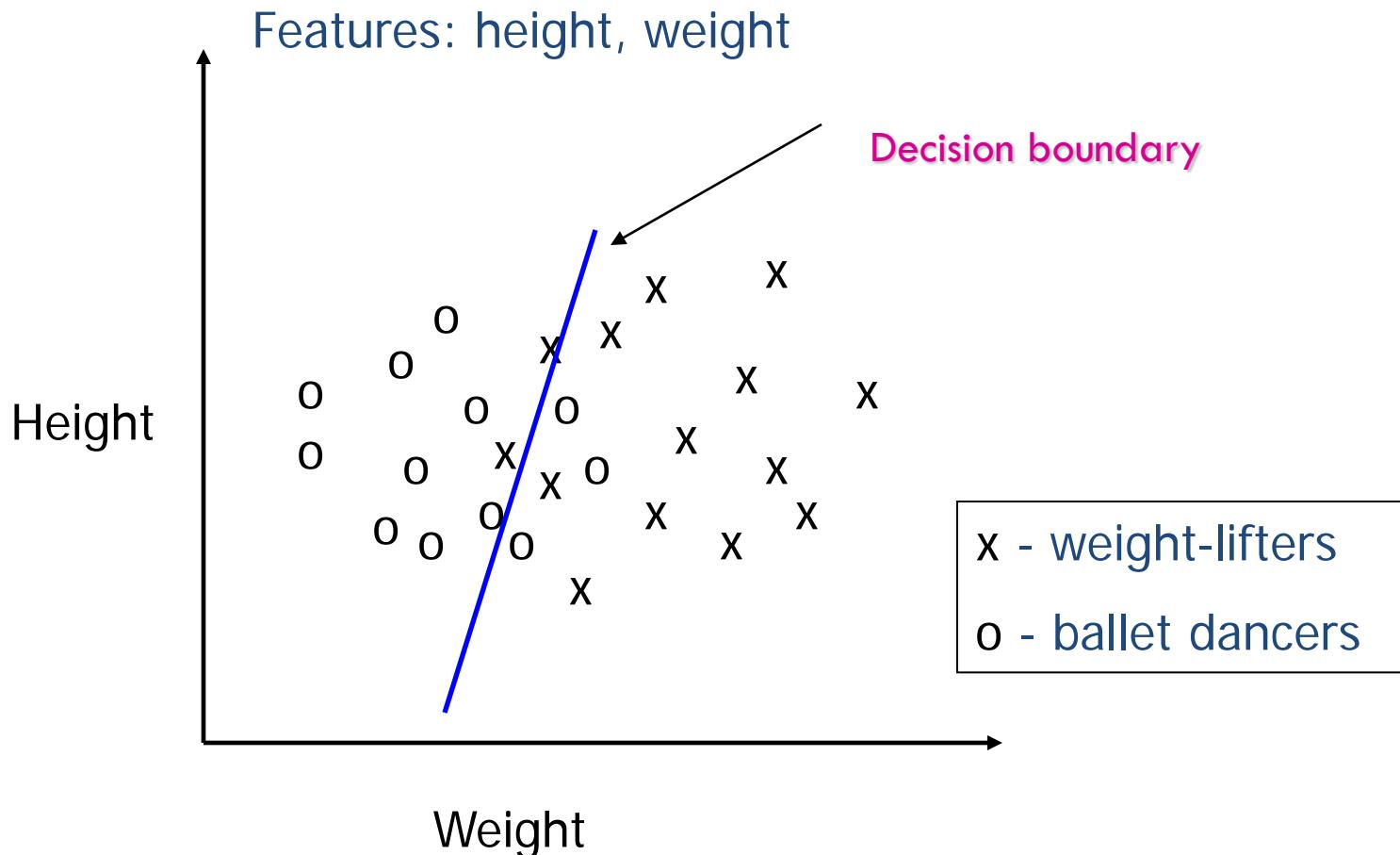
Sex	Age	Marital status	Net Income	...	Loan
F	28	Married	44K	.	?
M	47	Divorced	95K	.	?
F	30	Single	45K	.	?
M	55	Single	69K	.	?
M	45	Married	41K	.	?



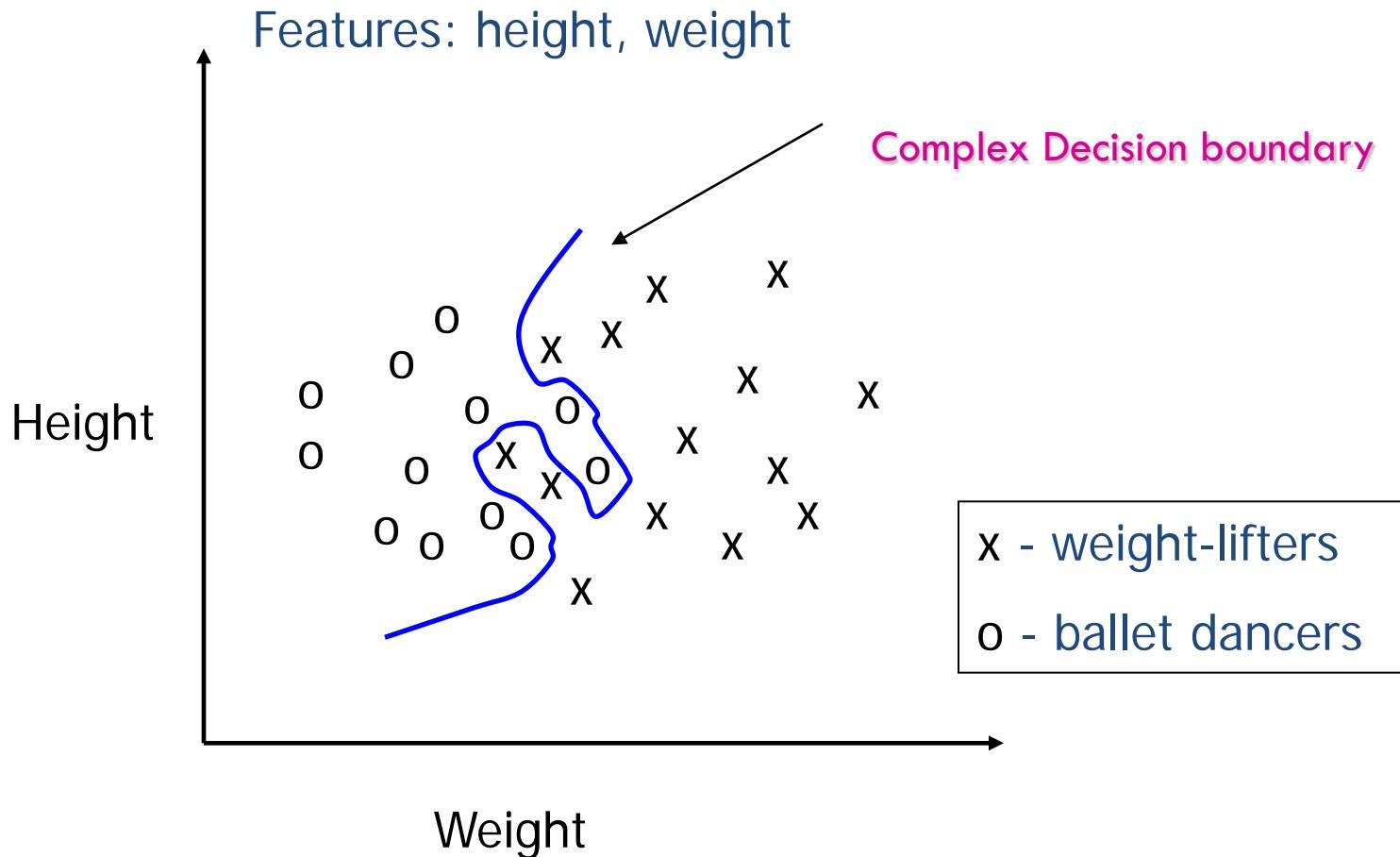
Classification example



Classification example - Simple Model



Classification example - Complex model



Note: A simple decision boundary is better than a complex one - It GENERALIZES better.

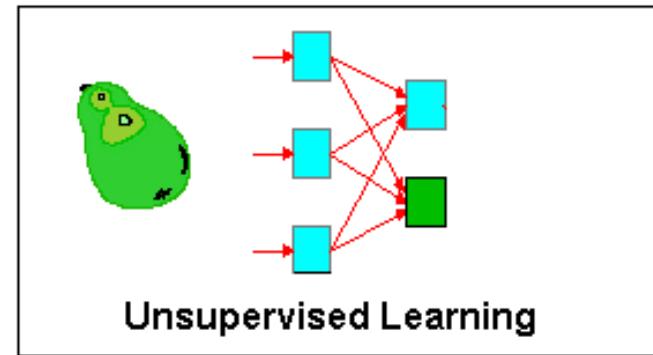
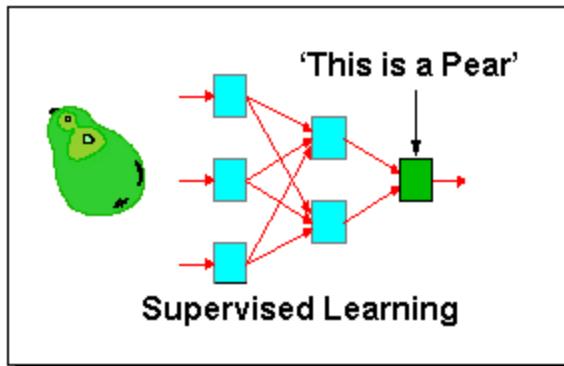
Machine Learning Methods

- Artificial Neural Networks
- Decision Trees
- Instance Based Methods (CBR, k-NN)
- Bayesian Networks
- Evolutionary Strategies
- Support Vector Machines
- ..

Learning Paradigms

The three main paradigms of machine learning are:

- Supervised learning - with teacher
 - inputs and correct outputs are provided by the teacher
- Reinforced learning - with reward or punishment
 - an action is evaluated
- Unsupervised learning - with no teacher
 - no hint about correct output is given



The Role of the teacher

- Supervised learning: the system uses a teacher
 - ▣ **Concept Learning:** teacher provides labeled data (pre-classified examples) to the system
 - ▣ **Reinforcement learning:** teacher provides an estimate of the quality of system's response to the data (e.g. positive/negative or scaled)
- Unsupervised learning: no teacher is available to the system
 - ▣ **Clustering:** partitioning or conceptual, flat or hierarchical
 - ▣ **Finding regularities in data:** Data Mining, Knowledge discovery

What does the system learn?

- **Prediction:** learning to predict values of unknown function
 - Classification: binary function
 - Regression: continuous-valued function
- **Concept learning:** the systems acquires descriptions of concepts
- **Explanation-based learning:** using traces (explanations) of correct (or incorrect) performances the system learns rules for more efficient performance of unseen tasks
- **Case-based (exemplar-based) learning:** the system memorizes cases (exemplars) of correctly classified data or correct performances and learns how to use them (e.g. by making analogies) to process unseen data

Reading assignment

- Why is learning hard?
- Write short notes on the following machine learning methods:
 - Artificial Neural Networks
 - Decision Trees
 - k-Nearest Neighbor

COMP 308

ARTIFICIAL INTELLIGENCE

PART 8.1 – LEARNING

Njeri Ireri

Jan – April 2020

Overview

- Introduction to machine learning
- Machine Learning Models
- Machine Learning Methods
- Machine Learning Paradigms

Learning & Adaptation

- Learning in general:
"Modification of a behavioral tendency by expertise."
(Webster 1984)
- Learning Machine:
"A learning machine, broadly defined is any device whose actions are influenced by past experiences." (Nilsson 1965)
- Machine Learning: (how does a machine learn – slide 3)
"Any change in a system that allows it to perform better the second time on repetition of the same task or on another task drawn from the same population." (Simon 1983)
"An improvement in information processing ability that results from information processing activity." (Tanimoto 1990)

Machine Learning



- Machine learning involves automatic procedures that learn a task from a series of examples
- Most convenient source of examples is **data**

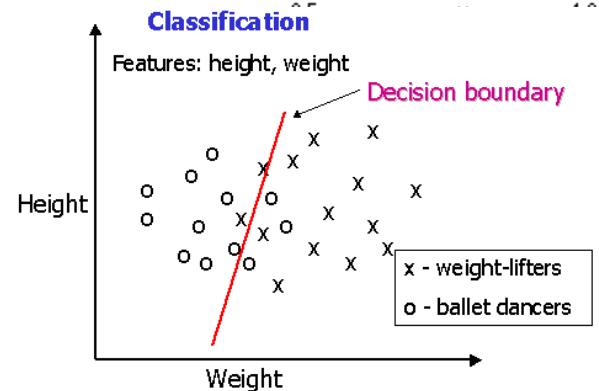
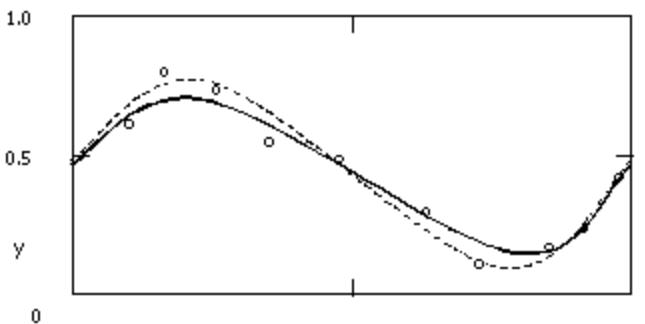
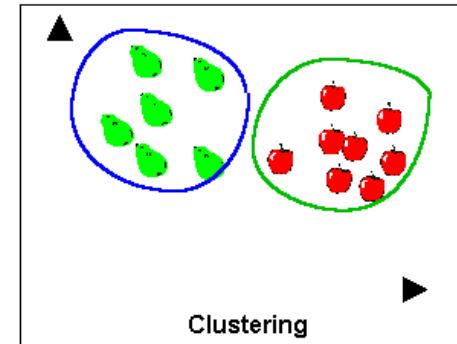
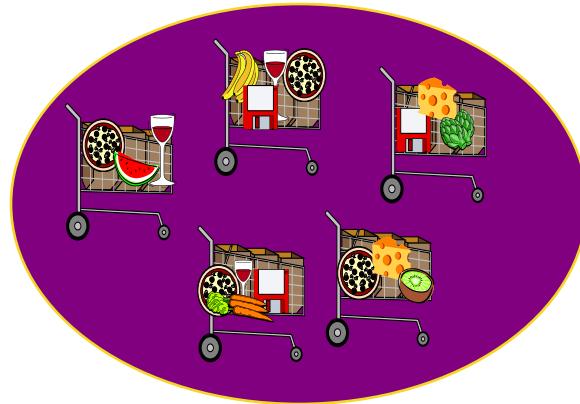
Learning

Definition:

A computer program is said to **learn** from **experience E** with respect to some class of **tasks T** and **performance measure P**, if its performance at tasks in T, as measured by P, improves with experience.

Machine Learning Models

- Classification
- Regression
- Clustering
- Time series analysis
- Association Analysis
- Sequence Discovery
-

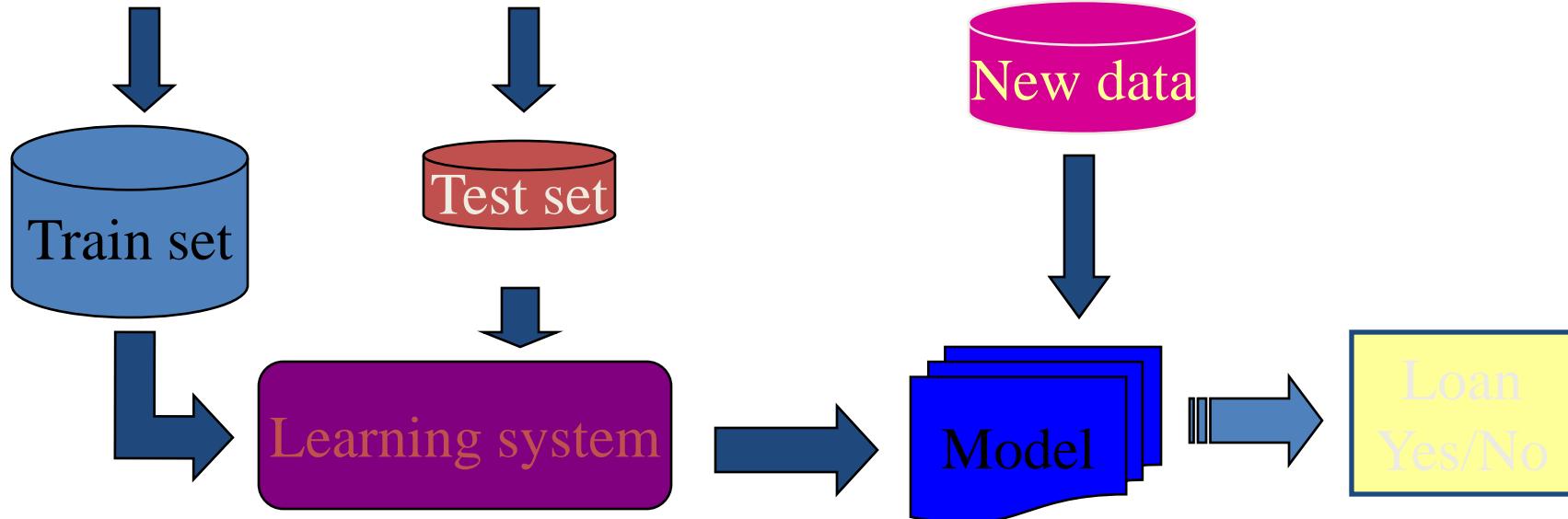


Classification example

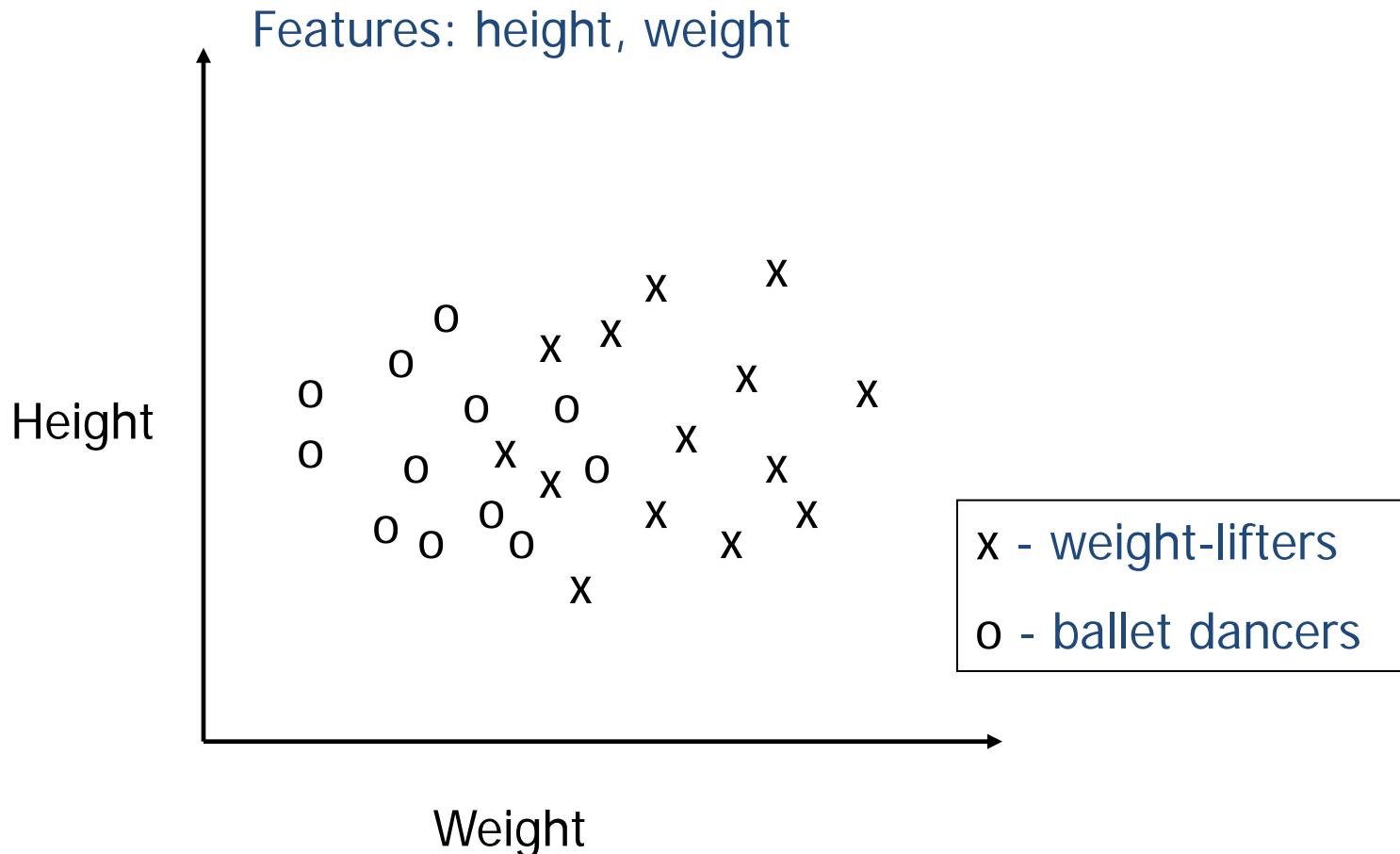
Loan Application Appraisal

No	Sex	Age	Marital status	Net Income	...	Loan
1	F	38	Married	45K	.	Yes
2	M	42	Married	66K	.	Yes
3	F	52	Single	43K	.	No
4	M	50	Single	70K	.	No
5	F	27	Married	40K	.	No
6	M	45	Divorced	38K	.	No
7	F	35	Widow	59K	.	Yes
8	M	32	Married	52K	.	Yes
..						

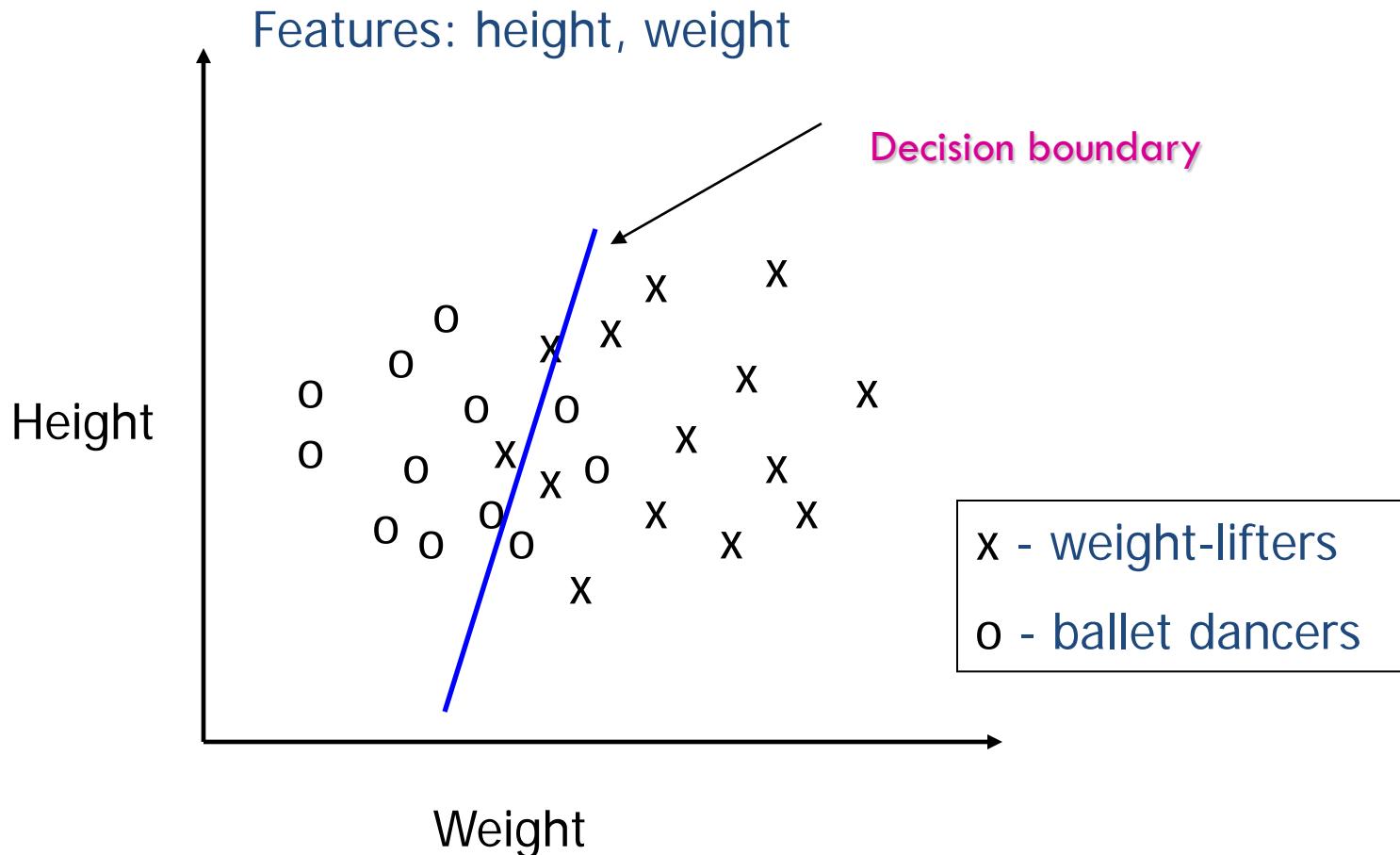
Sex	Age	Marital status	Net Income	...	Loan
F	28	Married	44K	.	?
M	47	Divorced	95K	.	?
F	30	Single	45K	.	?
M	55	Single	69K	.	?
M	45	Married	41K	.	?



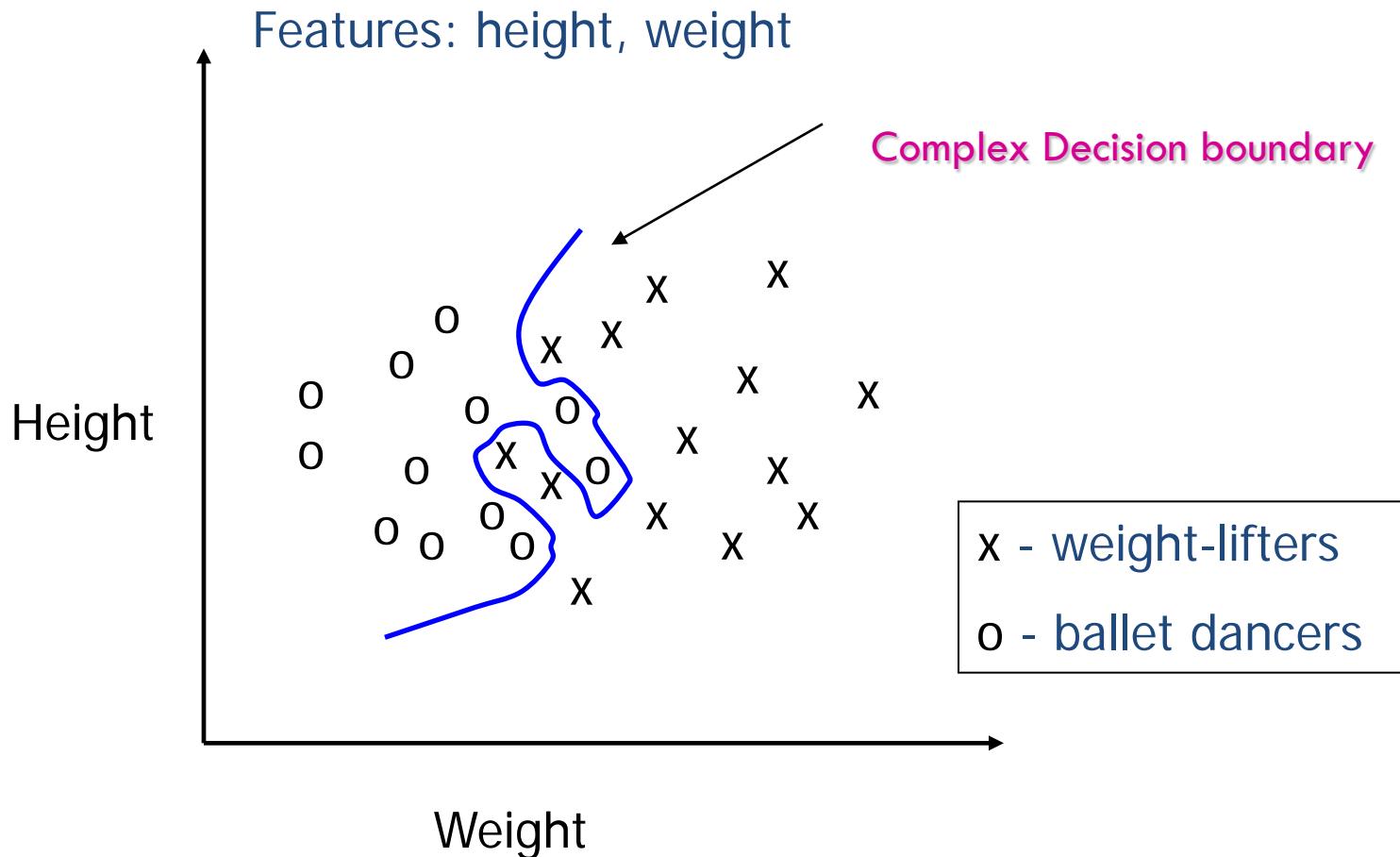
Classification example



Classification example - Simple Model



Classification example - Complex model



Note: A simple decision boundary is better than a complex one - It GENERALIZES better.

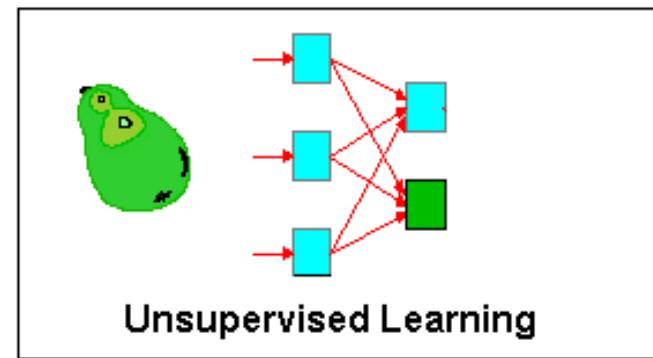
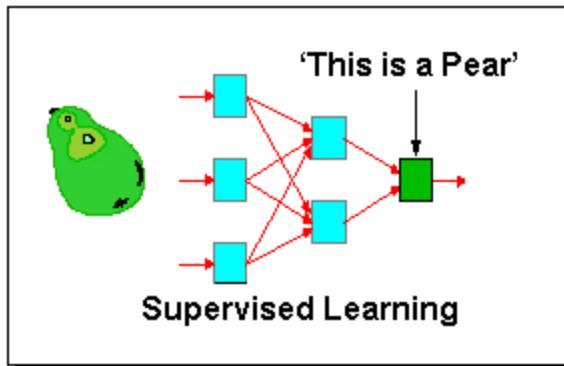
Machine Learning Methods

- Artificial Neural Networks
- Decision Trees
- Instance Based Methods (CBR, k-NN)
- Bayesian Networks
- Evolutionary Strategies
- Support Vector Machines
- ..

Learning Paradigms

The three main paradigms of machine learning are:

- Supervised learning - with teacher
 - inputs and correct outputs are provided by the teacher
- Reinforced learning - with reward or punishment
 - an action is evaluated
- Unsupervised learning - with no teacher
 - no hint about correct output is given



The Role of the teacher

- Supervised learning: the system uses a teacher
 - ▣ **Concept Learning:** teacher provides labeled data (pre-classified examples) to the system
 - ▣ **Reinforcement learning:** teacher provides an estimate of the quality of system's response to the data (e.g. positive/negative or scaled)
- Unsupervised learning: no teacher is available to the system
 - ▣ **Clustering:** partitioning or conceptual, flat or hierarchical
 - ▣ **Finding regularities in data:** Data Mining, Knowledge discovery

What does the system learn?

- **Prediction:** learning to predict values of unknown function
 - Classification: binary function
 - Regression: continuous-valued function
- **Concept learning:** the systems acquires descriptions of concepts
- **Explanation-based learning:** using traces (explanations) of correct (or incorrect) performances the system learns rules for more efficient performance of unseen tasks
- **Case-based (exemplar-based) learning:** the system memorizes cases (exemplars) of correctly classified data or correct performances and learns how to use them (e.g. by making analogies) to process unseen data

Reading assignment

- Why is learning hard?

COMP 308

ARTIFICIAL INTELLIGENCE

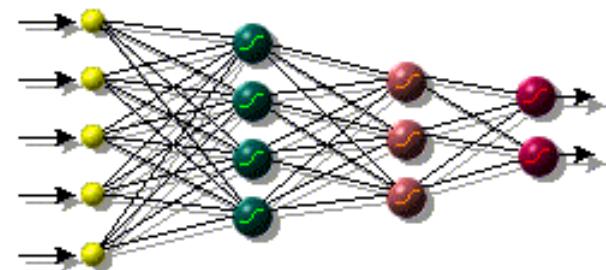
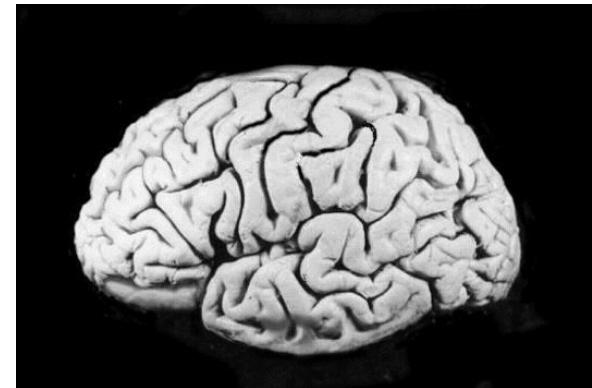
PART 8.2 – ARTIFICIAL NEURAL NETWORKS

Njeri Ireri

Jan – April 2020

Overview

- The Brain
- Brain vs. Computers
- The Perceptron
- Multilayer networks
- Some Applications



Artificial Neural Networks

□ Other terms/names

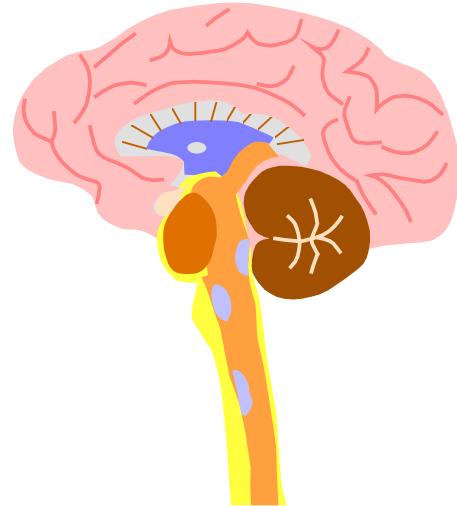
- connectionist
- parallel distributed processing
- neural computation
- adaptive networks..

□ History

- 1943-McCulloch & Pitts are generally recognised as the designers of the first neural network
- 1949-First learning rule
- 1969-Minsky & Papert - perceptron limitation - Death of ANN
- 1980's - Re-emergence of ANN - multi-layer networks

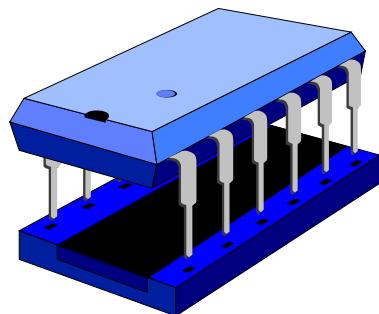
Brain and Machine

- The Brain
 - Pattern Recognition
 - Association
 - Complexity
 - Noise Tolerance

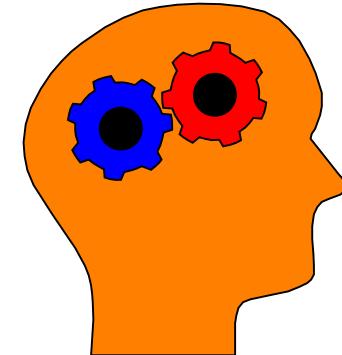


- The Machine
 - Calculation
 - Precision
 - Logic

The contrast in architecture



- The **Von Neumann architecture** uses a single processing unit;
 - Tens of millions of operations per second
 - Absolute arithmetic precision
- **The brain** uses many slow unreliable processors acting in **parallel**

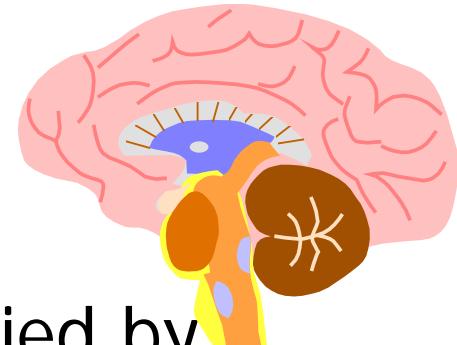


Features of the Brain



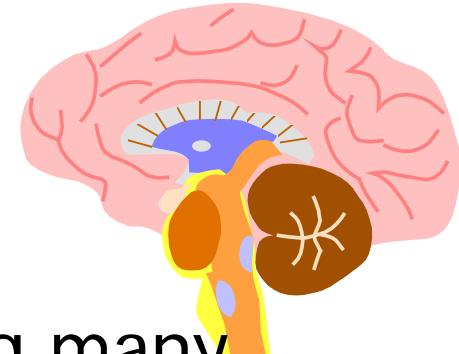
- Ten billion (10^{10}) neurons
- On average, several thousand connections
- Hundreds of operations per second
- Die off frequently (never replaced)
- Compensates for problems by massive parallelism

The biological inspiration



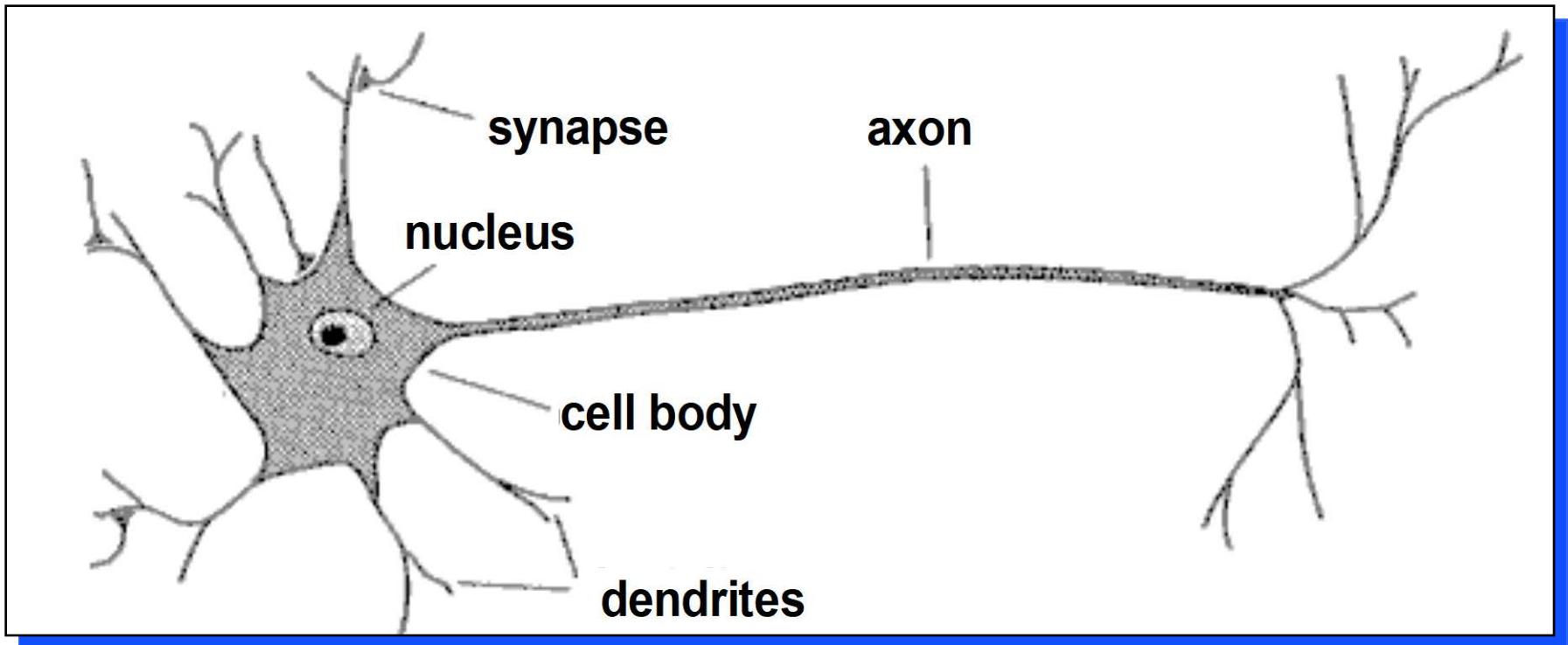
- The brain has been extensively studied by scientists.
- Vast complexity prevents all but rudimentary understanding.
- Even the behaviour of an individual neuron is extremely complex

The biological inspiration



- Single “percepts” distributed among many neurons
- Localized parts of the brain are responsible for certain well-defined functions (e.g. vision, motion).
- Which features are integral to the brain's performance?
- Which are incidentals imposed by the fact of biology?

The Structure of Neurons



The Structure of Neurons



The Structure of Neurons

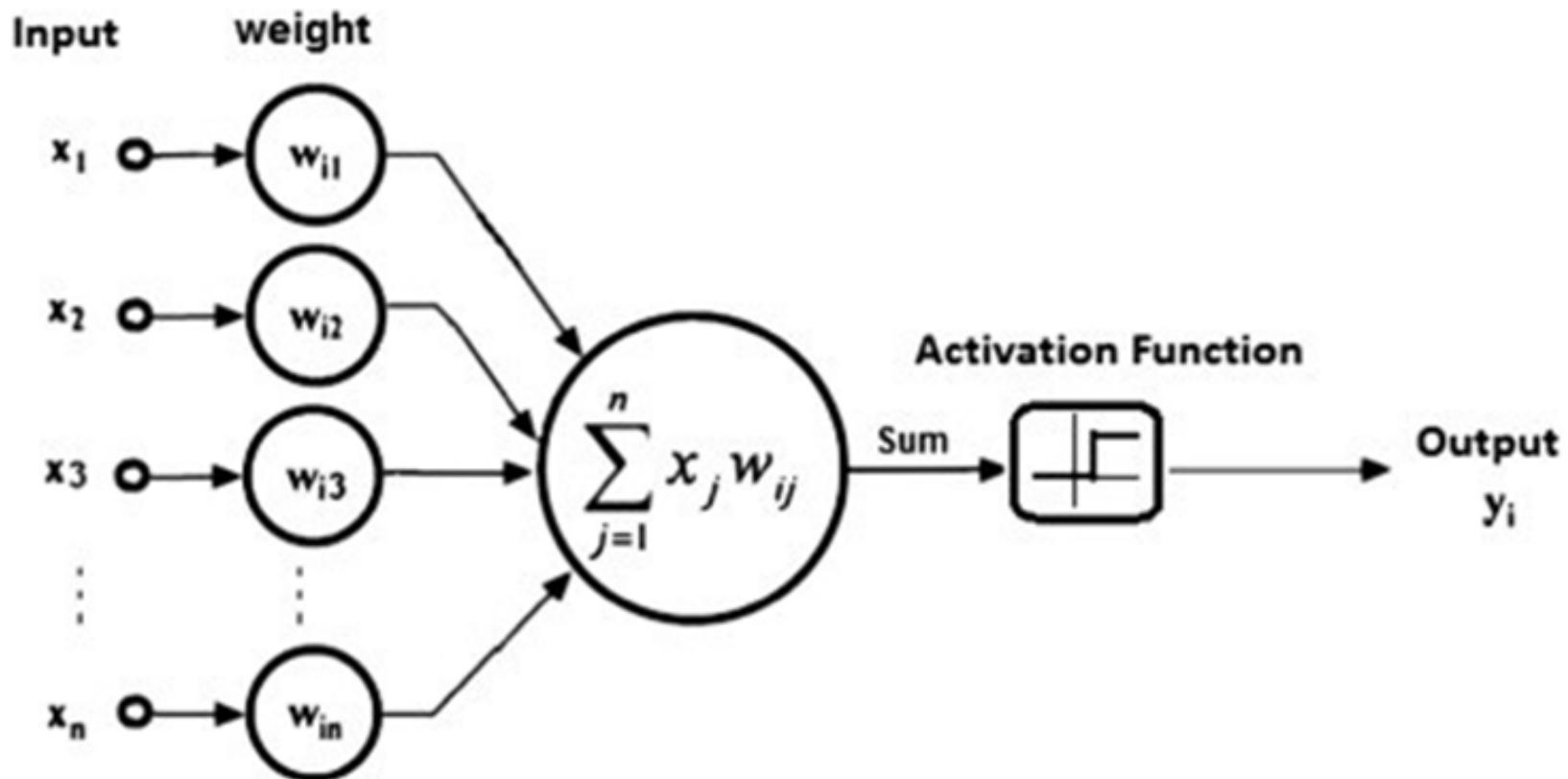
A neuron has a cell body, a branching input structure (the dendrite) and a branching output structure (the axon)

- Axons connect to dendrites via synapses
- Electro-chemical signals are propagated from the dendritic input, through the cell body, and down the axon to other neurons

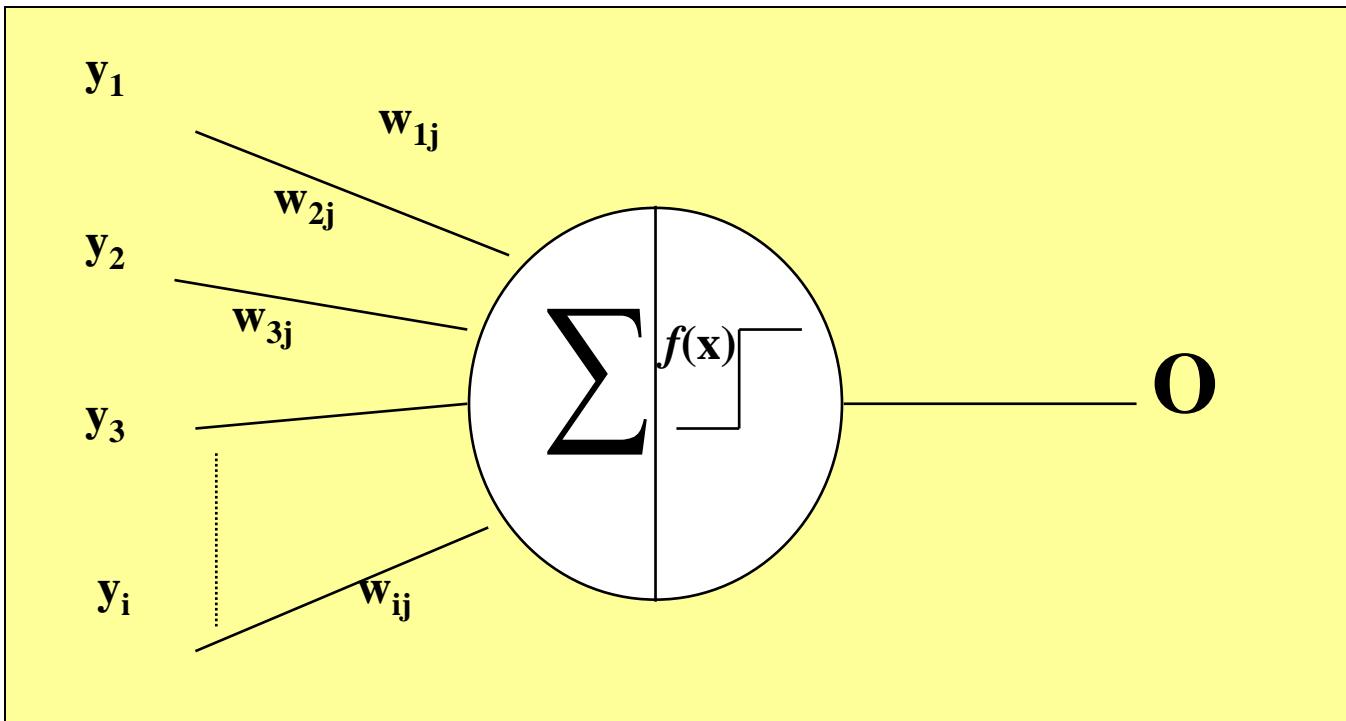
The Structure of Neurons

- A neuron only fires if its input signal exceeds a certain amount (the **threshold**) in a short time period.
- Synapses vary in strength
 - Good connections allowing a large signal
 - Slight connections allow only a weak signal
 - Synapses can be either **excitatory** or **inhibitory**.

The Artificial Neuron (Perceptron)



A Simple Model of a Neuron (Perceptron)



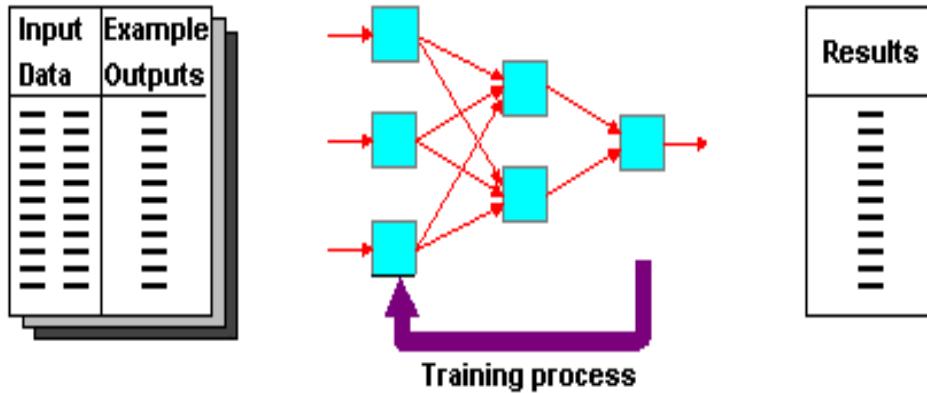
A Simple Model of a Neuron

- how it works

- Each neuron has weighted inputs from other neurons and the input signals form a weighted sum
- Each neuron also has a threshold value and an activation function which transforms neuron's input into output as follows:
 - ▣ The unit performs a weighted sum of its inputs, and subtracts its threshold value, to give its activation level
 - ▣ Activation level is passed through a sigmoid activation function and if it exceeds the threshold, the neuron “fires”, meaning that an output signal is given by the neuron

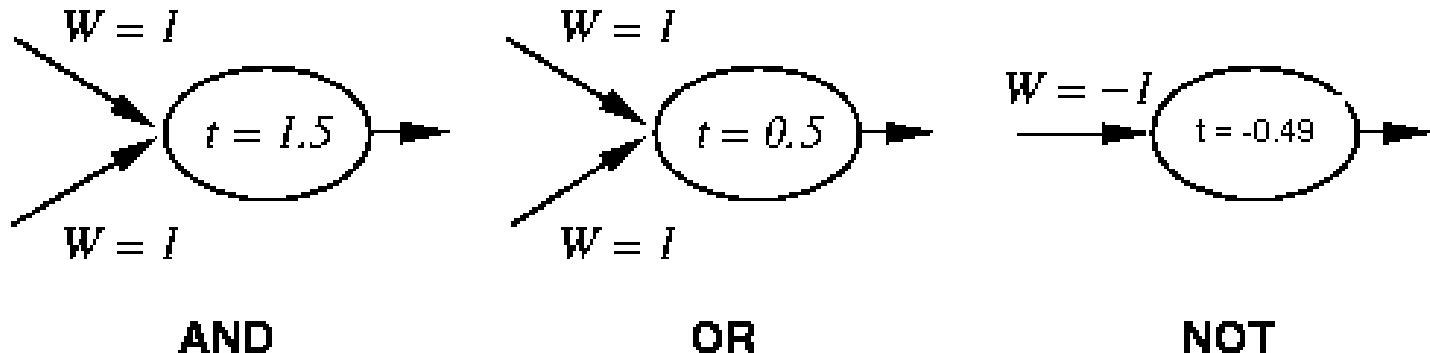
Supervised Learning

- Training and test data sets
- Training set; input & target



Sepal length	Sepal width	Petal length	Petal width	Class
5.1	3.5	1.4	0.2	0
4.9	3.0	1.4	0.2	2
4.7	3.2	1.3	0.2	0
4.6	3.1	1.5	0.2	1

Perceptron Training

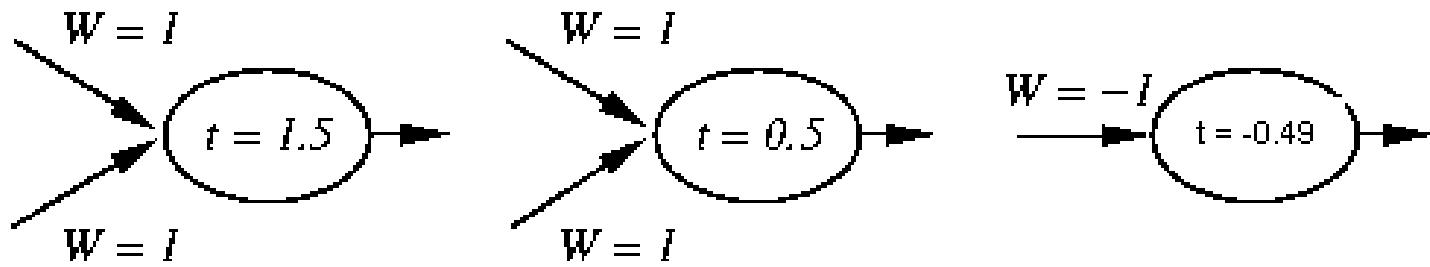


activation function

$$\text{Output} = \begin{cases} 1 & \text{if } \sum_{i=0} w_i x_i > t \\ 0 & \text{otherwise} \end{cases}$$

- Linear threshold is used.
- W - weight value
- t - threshold value

Simple network



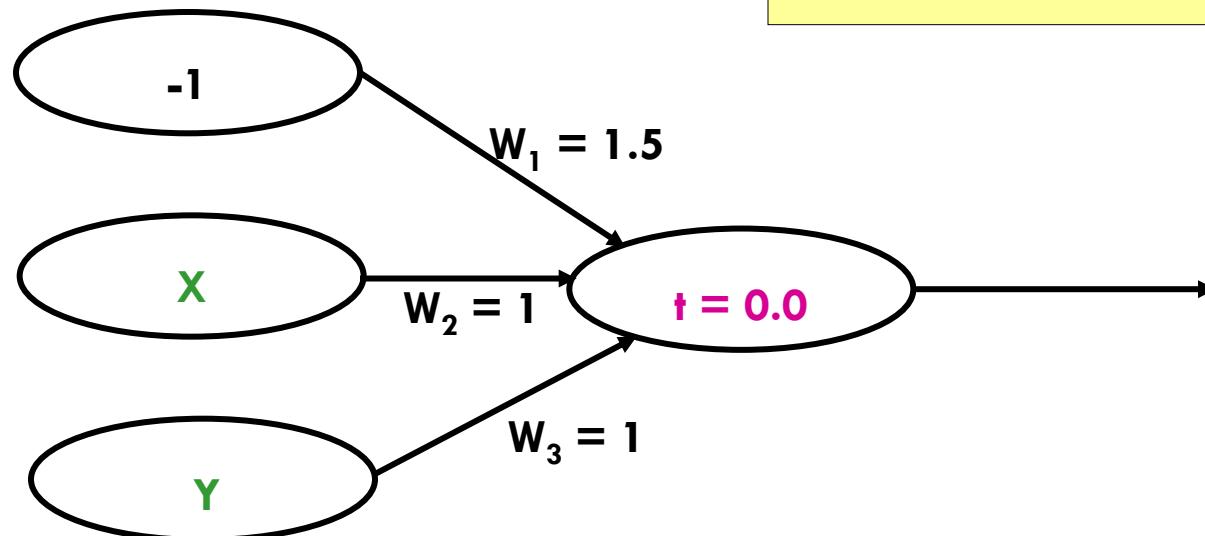
AND

OR

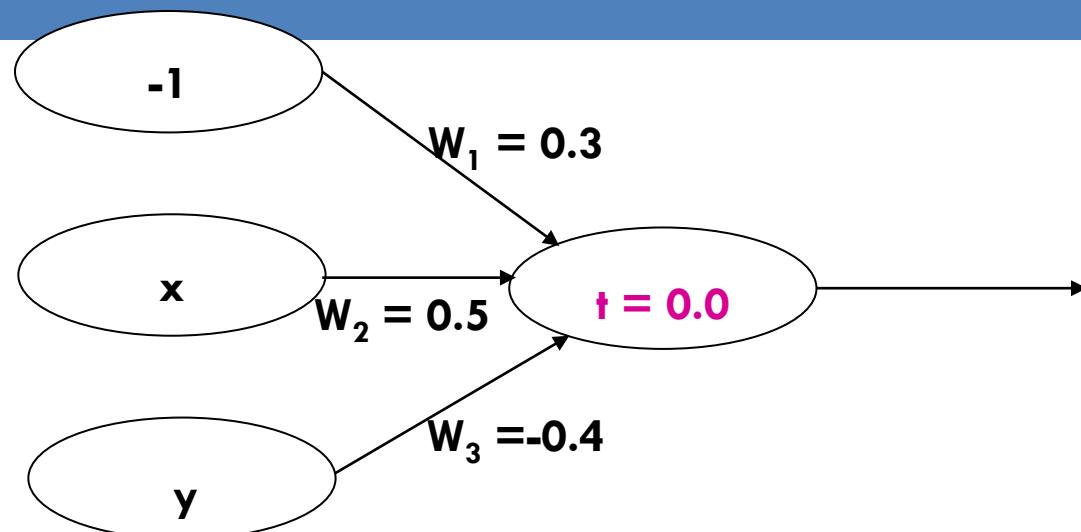
NOT

AND with a Biased input

$$\text{output} = \begin{cases} 1 & \text{if } \sum_{i=0} w_i x_i > t \\ 0 & \text{otherwise} \end{cases}$$



Training Perceptrons

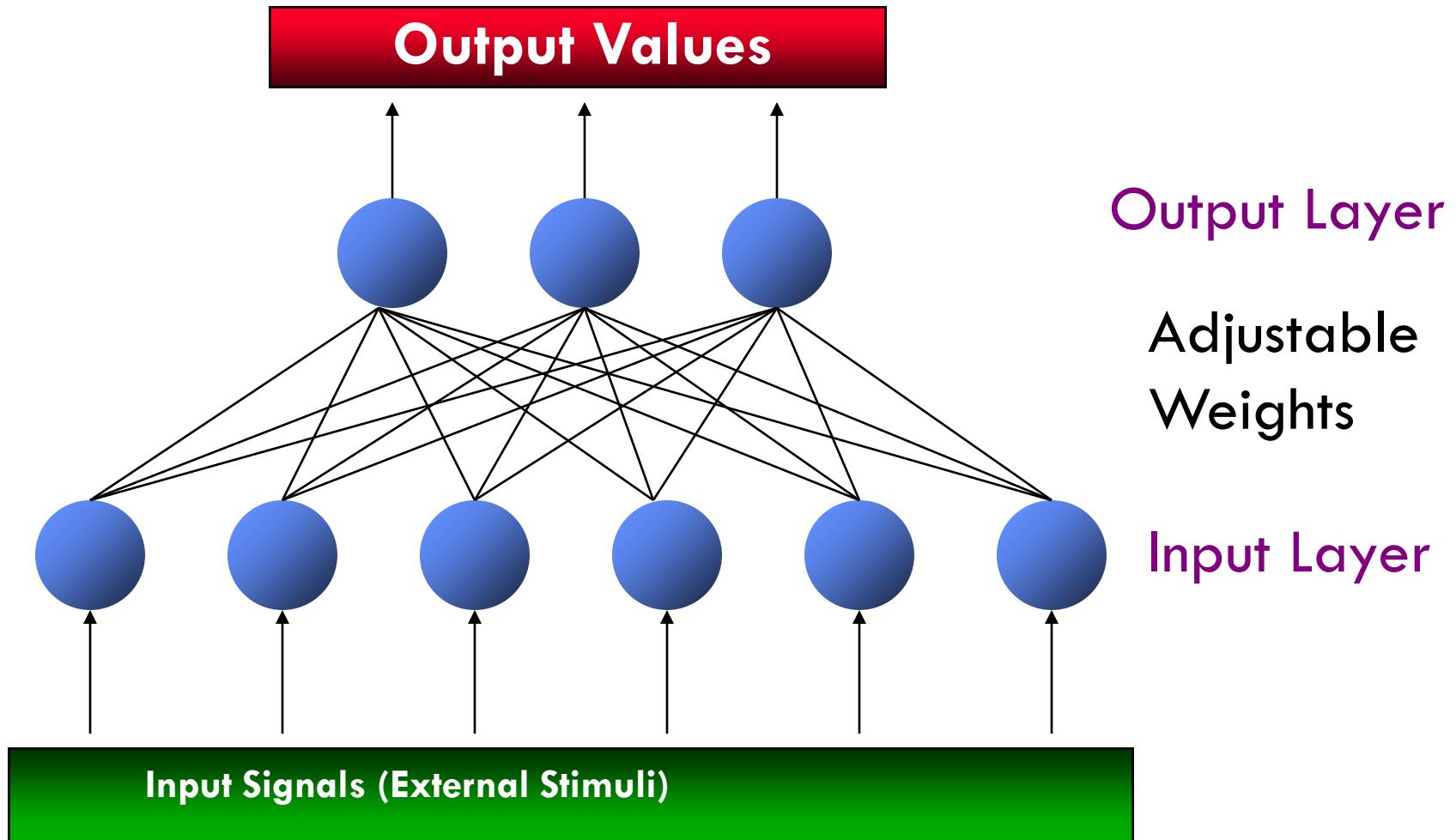


For AND

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

I_1	I_2	I_3	Summation	Output
-1	0	0	$(-1*0.3) + (0*0.5) + (0*-0.4) = -0.3$	0
-1	0	1	$(-1*0.3) + (0*0.5) + (1*-0.4) = -0.7$	0
-1	1	0	$(-1*0.3) + (1*0.5) + (0*-0.4) = 0.2$	1
-1	1	1	$(-1*0.3) + (1*0.5) + (1*-0.4) = -0.2$	0

Multilayer Perceptron (MLP)



Types of Layers

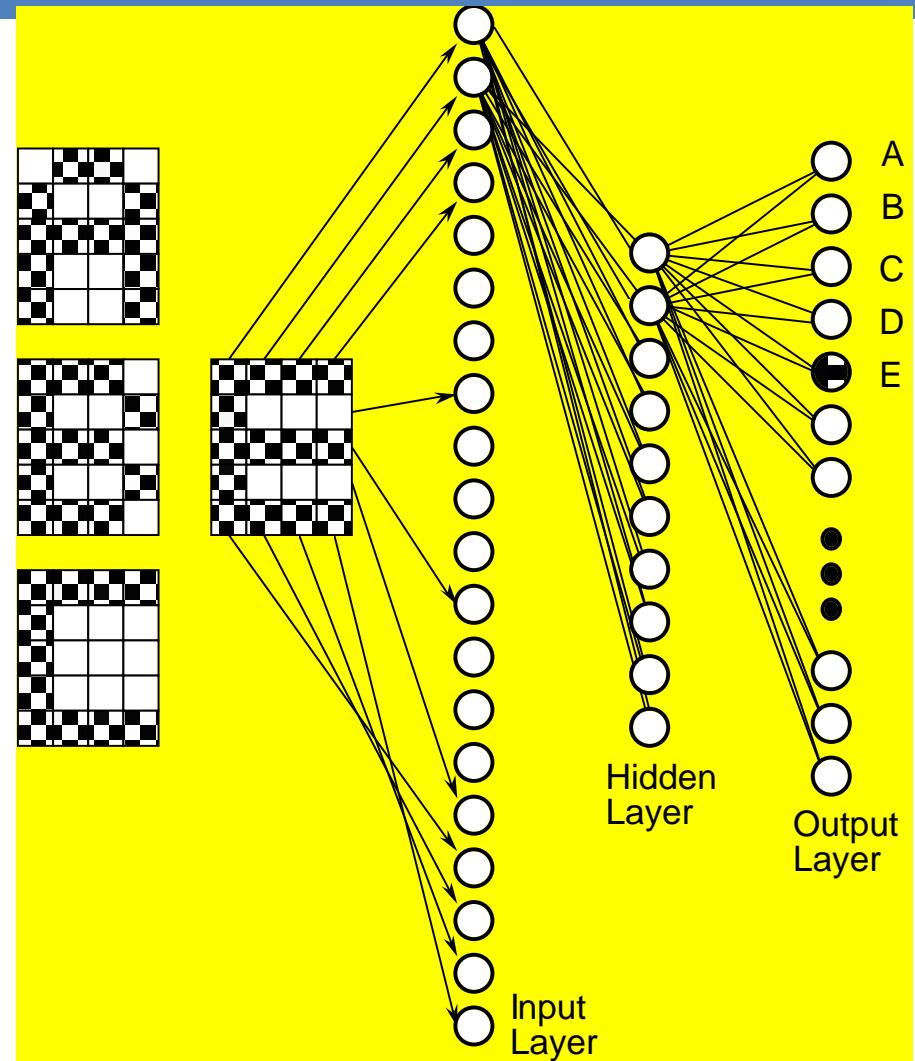
- The input layer.
 - Introduces input values into the network.
 - No activation function or other processing.
- The hidden layer(s).
 - Perform classification of features
 - Two hidden layers are sufficient to solve any problem
 - Features imply more layers may be better(There can be more than one hidden layers which are used for processing the inputs received from the input layers)
- The output layer.
 - Functionally just like the hidden layers
 - Outputs are passed on to the world outside the neural network.

Properties of neural networks

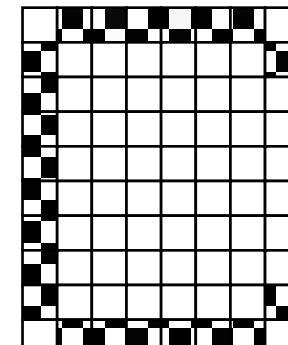
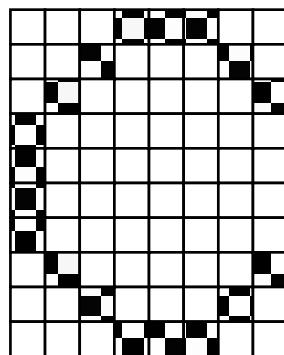
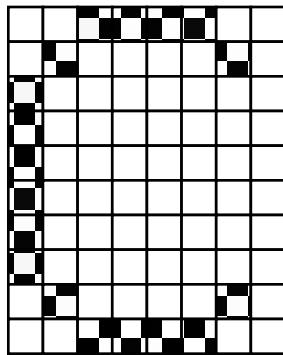
- The properties of neural networks define where they are useful.
 - Can learn complex mappings from inputs to outputs, based solely on samples
 - Difficult to analyse: firm predictions about neural network behaviour difficult;
 - Unsuitable for safety-critical applications.
 - Require limited understanding from trainer, who can be guided by heuristics.
 - **Application** - The properties of neural networks define where they are useful

Neural network for OCR

- OCR - optical character recognition
- feedforward network
- trained using Back-propagation



OCR for 8x10 characters



10

8

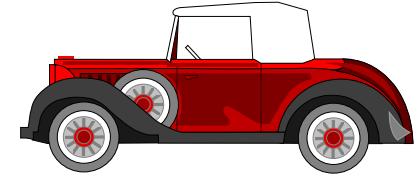
8

10

10

- NN are able to generalise
- learning involves generating a partitioning of the input space
- for single layer network input space must be linearly separable
- what is the dimension of this input space?
- how many points in the input space?
- this network is binary(uses binary values)
- networks may also be continuous

Engine management



- The behaviour of a car engine is influenced by a large number of parameters
 - temperature at various points
 - fuel/air mixture
 - lubricant viscosity.
- Major companies have used neural networks to dynamically tune an engine depending on current settings.

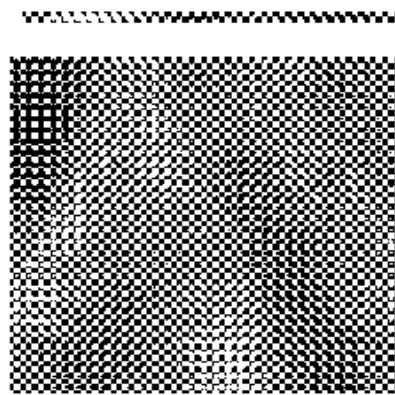
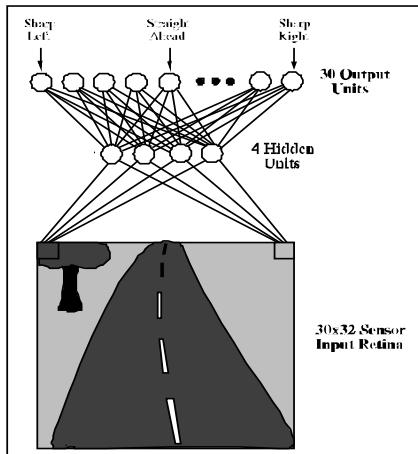
ALVINN

Drives 70 mph on a public highway

30 outputs
for steering

4 hidden
units

30x32 pixels
as inputs

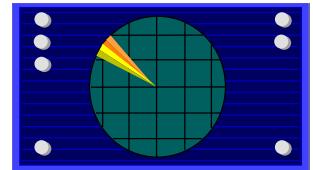


30x32 weights
into one out of
four hidden
unit

Signature recognition

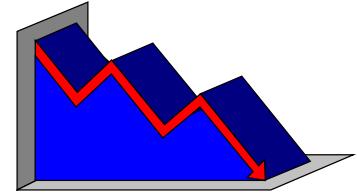
- Each person's signature is different.
- There are structural similarities which are difficult to quantify.
- One company has manufactured a machine which recognizes signatures to within a high level of accuracy.
 - Considers speed in addition to gross shape.
 - Makes forgery even more difficult.

Sonar target recognition



- Distinguish mines from rocks on sea-bed
- The neural network is provided with a large number of parameters which are extracted from the sonar signal.
- The training set consists of sets of signals from rocks and mines.

Stock market prediction



- “Technical trading” refers to trading based solely on known statistical parameters; e.g. previous price
- Neural networks have been used to attempt to predict changes in prices.
- Difficult to assess success since companies using these techniques are reluctant to disclose information.

Mortgage assessment



- Assess risk of lending to an individual.
- Difficult to decide on marginal cases.
- Neural networks have been trained to make decisions, based upon the opinions of expert underwriters.
- Neural network produced a 12% reduction in delinquencies compared with human experts.

Neural Network Problems

- Many Parameters to be set
- Overfitting
- long training times
- ...

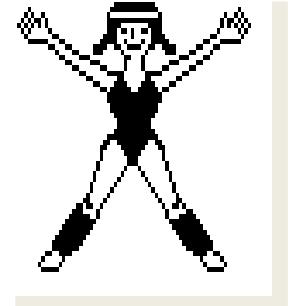
Parameter setting

- Number of layers
- Number of neurons
 - too many neurons, require more training time
- Learning rate
 - from experience, value should be small ~0.1
- Momentum term
- ..

Over-fitting

- With sufficient nodes can classify any training set exactly
- May have poor generalisation ability.
- Cross-validation with some patterns
 - Typically 30% of training patterns
 - Validation set error is checked each epoch
 - Stop training if validation error goes up

Training time



- How many epochs of training?
 - Stop if the error fails to improve (has reached a minimum)
 - Stop if the rate of improvement drops below a certain level
 - Stop if the error reaches an acceptable level
 - Stop when a certain number of epochs have passed

COMP 308

ARTIFICIAL INTELLIGENCE

PART 8.4 – K-NEAREST

NEIGHBOUR

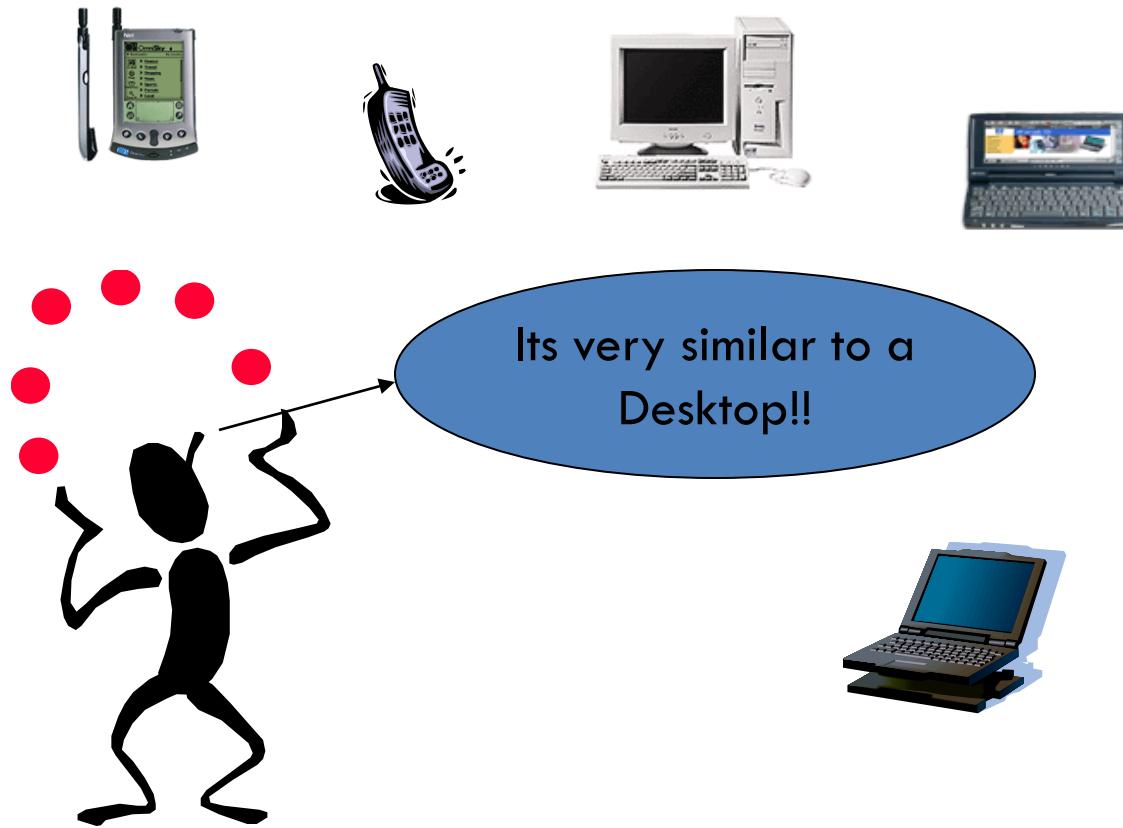
Njeri Ireri

Jan – April 2020

Overview

- Nearest Neighbor
- Locally weighted regression
- Case based reasoning (CBR)

Instance-based Learning



Nearest Neighbors

- **Nearest-neighbor** classifiers are based on learning by analogy, that is, by comparing a given test tuple with training tuples that are similar to it
- A **k-nearest-neighbor** often abbreviated K-NN, is an approach to data classification that estimates how likely a data point is to be a member of one group or the other depending on what group the data points *nearest* to it are in.
- Formally, the nearest-neighbor (NN) search problem is defined as follows:
 - given a set S of points in a space M and a query point $q \in M$, find the closest point in S to q

When to Consider Nearest Neighbors

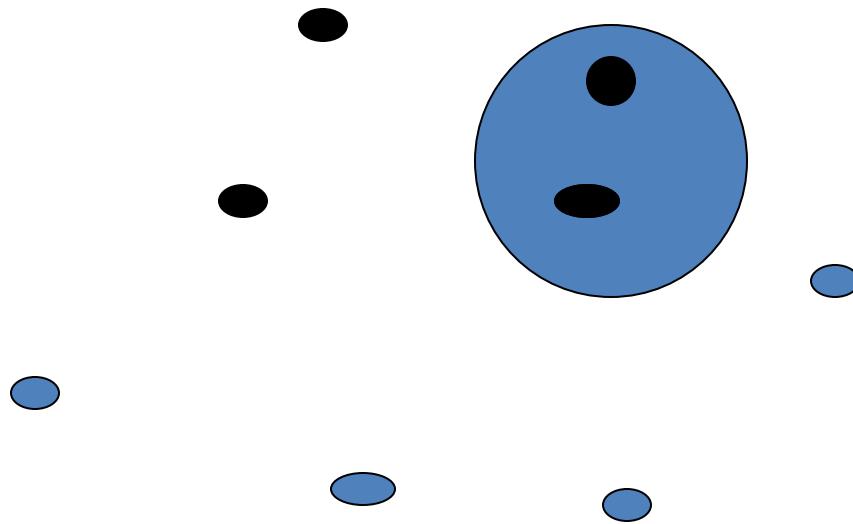
- Instances map to points in \mathbb{R}^n (where n-dimensional space)
- Less than 20 attributes per instance
- Lots of training data

Advantages :

These are the strengths of the k-Nearest Neighbour machine learning technique:

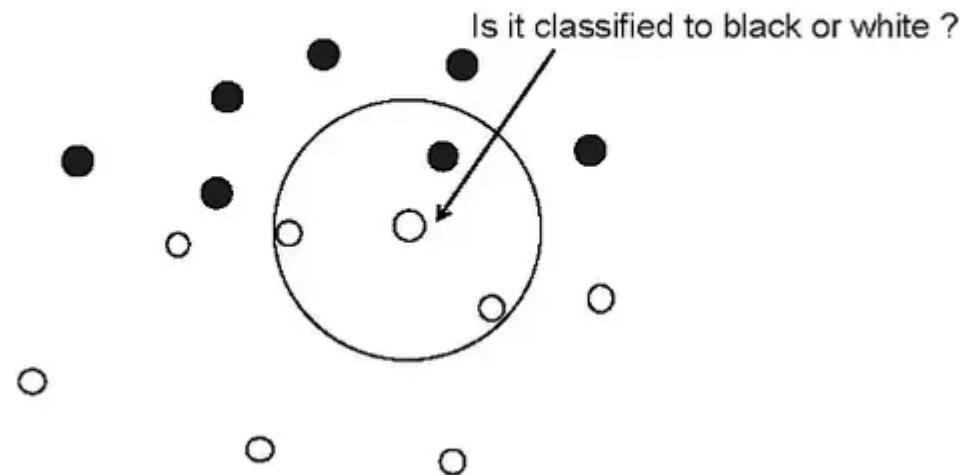
- has the ability to model very complex target functions by a collection of less complex approximations
- It is easy to program this algorithm
- No optimization or training is required for this algorithm

1-Nearest Neighbor

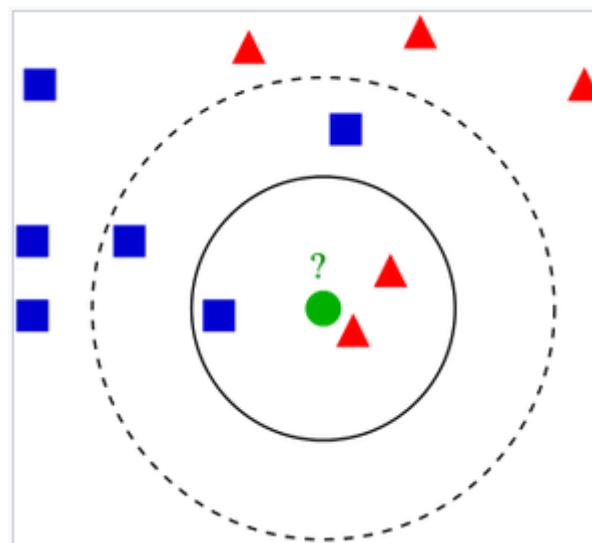


3 - Nearest Neighbors

3-Nearest Neighbor



Nearest Neighbors



Nearest Neighbors

- **Nearest neighbor search (NNS)**, as a form of proximity search, is the optimization problem of finding the point in a given set that is closest (or most similar) to a given point. Closeness is typically expressed in terms of a dissimilarity function: the less similar the objects, the larger the function values. Formally, the nearest-neighbor (NN) search problem is defined as follows: given a set S of points in a space M and a query point $q \in M$, find the closest point in S to q .
- **Nearest Neighbor Analysis** is a method for classifying cases based on their similarity to other cases. In machine learning, it was developed as a way to recognize patterns of data without requiring an exact match to any stored patterns, or cases. Nearest neighbor analysis examines the distances between each point and the closest point to it, and then compares these to expected values for a random sample of points from a CSR (complete spatial randomness) pattern. CSR is generated by means of two assumptions: 1) that all places are equally likely to be the recipient of a case (event) and 2) all cases are located independently of one another.

When to Consider Nearest Neighbors

- It is robust to noisy training data. Provided with sufficiently large training dataset, it has been shown to be quite effective.
- Information can be incrementally added at run-time
- Information is never lost because the training examples are stored explicitly

Disadvantages:

- Slow at query time
- Easily fooled by irrelevant attributes

Instance Based Learning

Key idea: just store all training examples $\langle x_i, f(x_i) \rangle$

Nearest neighbor:

- Given query instance x_q , first locate nearest training example \hat{x}_n , then estimate $f(x_q) = f(\hat{x}_n)$

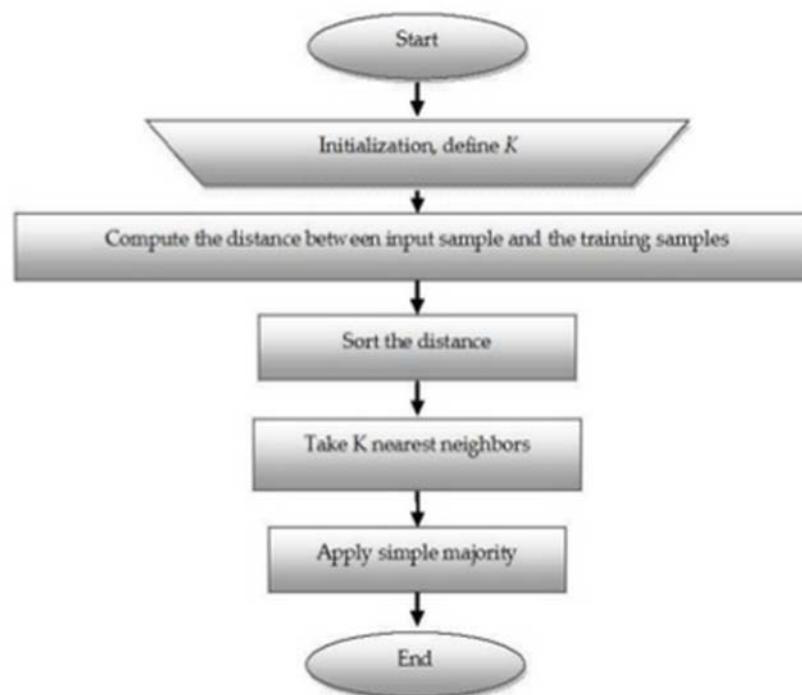
K-nearest neighbor:

- Given x_q , take vote among its k nearest neighbors (if discrete-valued target function)
- Take mean of f values of k nearest neighbors (if real-valued) $f(x_q) = \hat{\sum}_{i=1}^k f(x_i) / k$

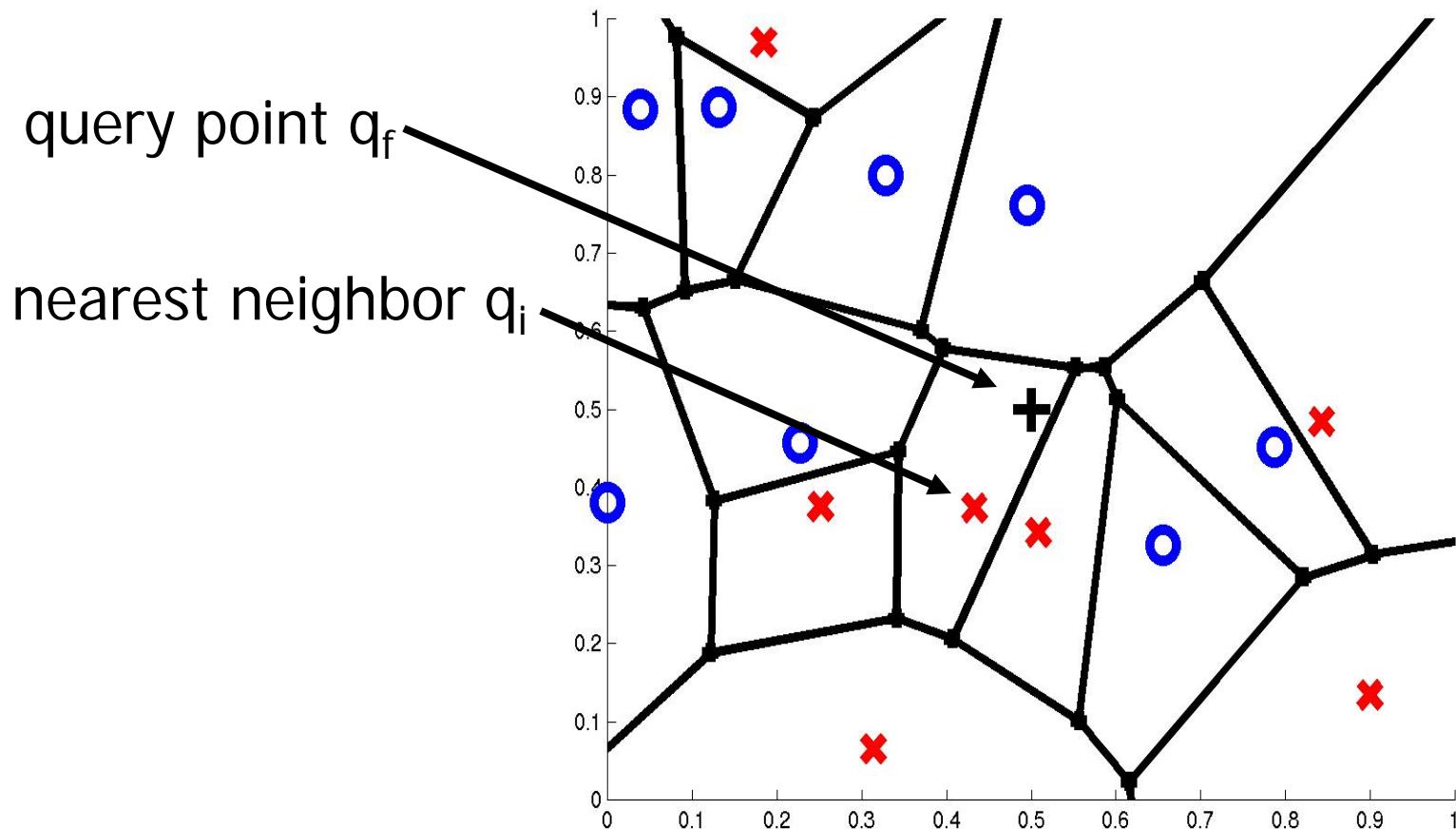
Note: \hat{f} - value returned by algorithm

KNN classifier Algorithm

KNN Classifier Algorithm

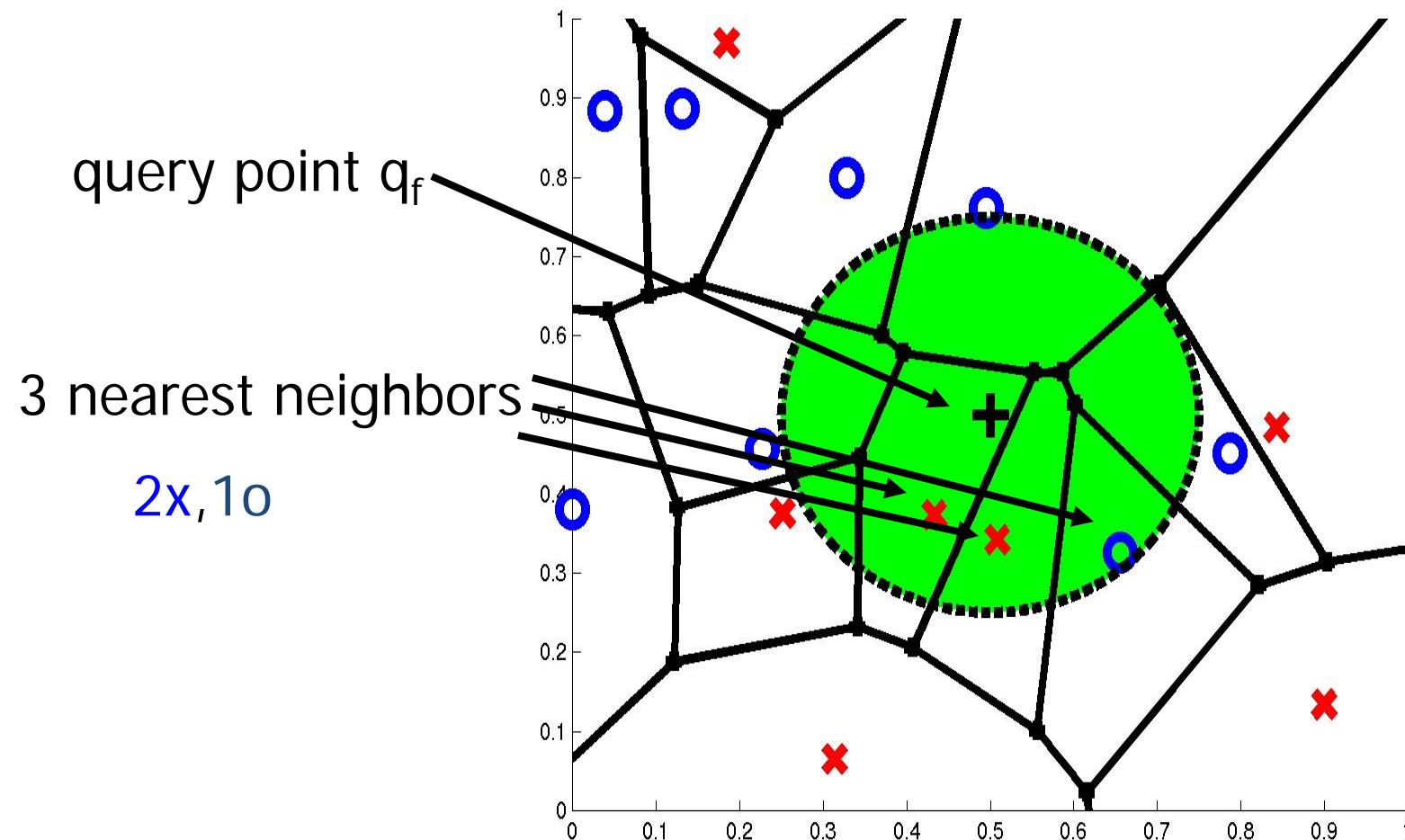


Voronoi Diagram

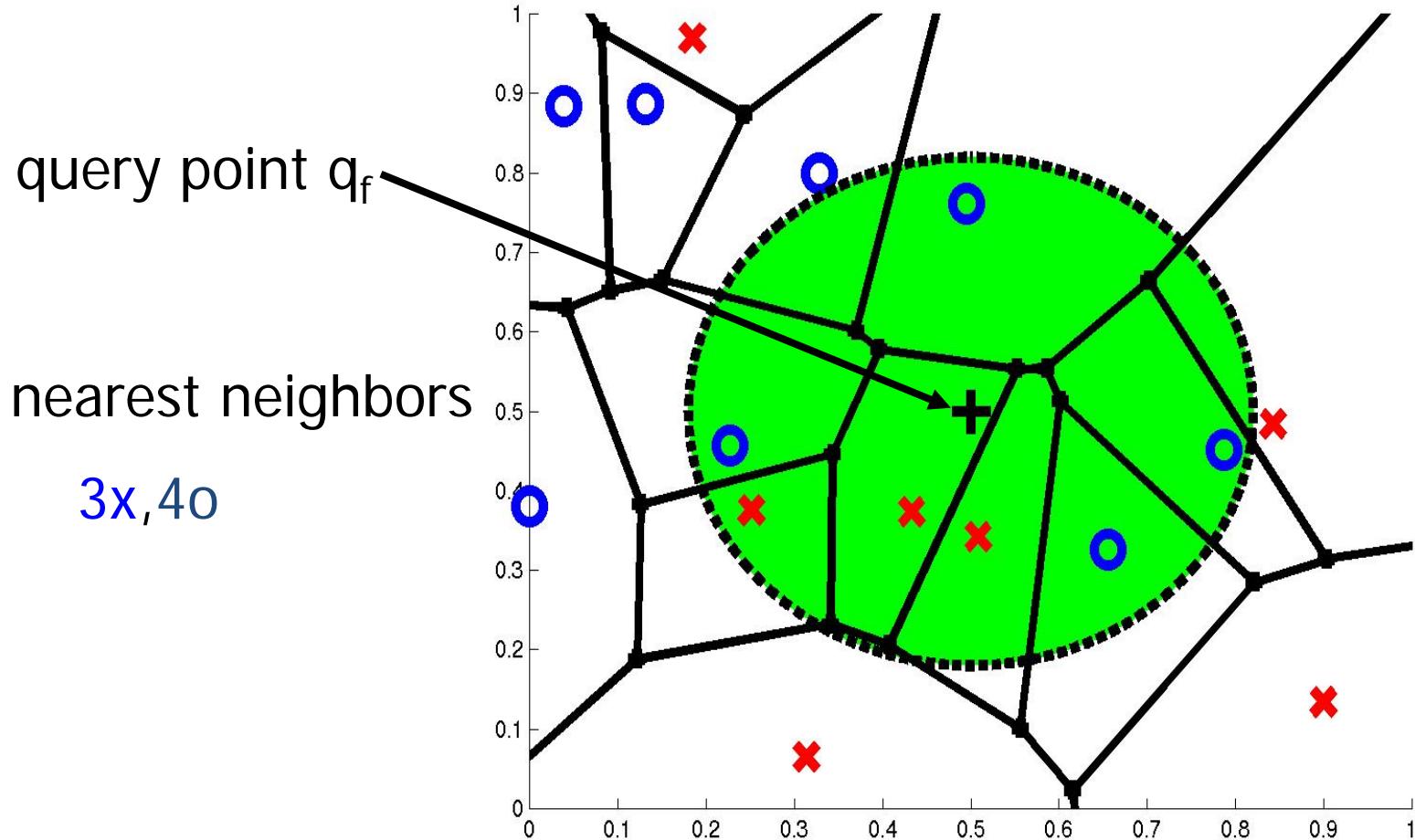


Example: complex 2-class problem (\times, \circ)
using 2 attributes on x and y axis

3-Nearest Neighbors

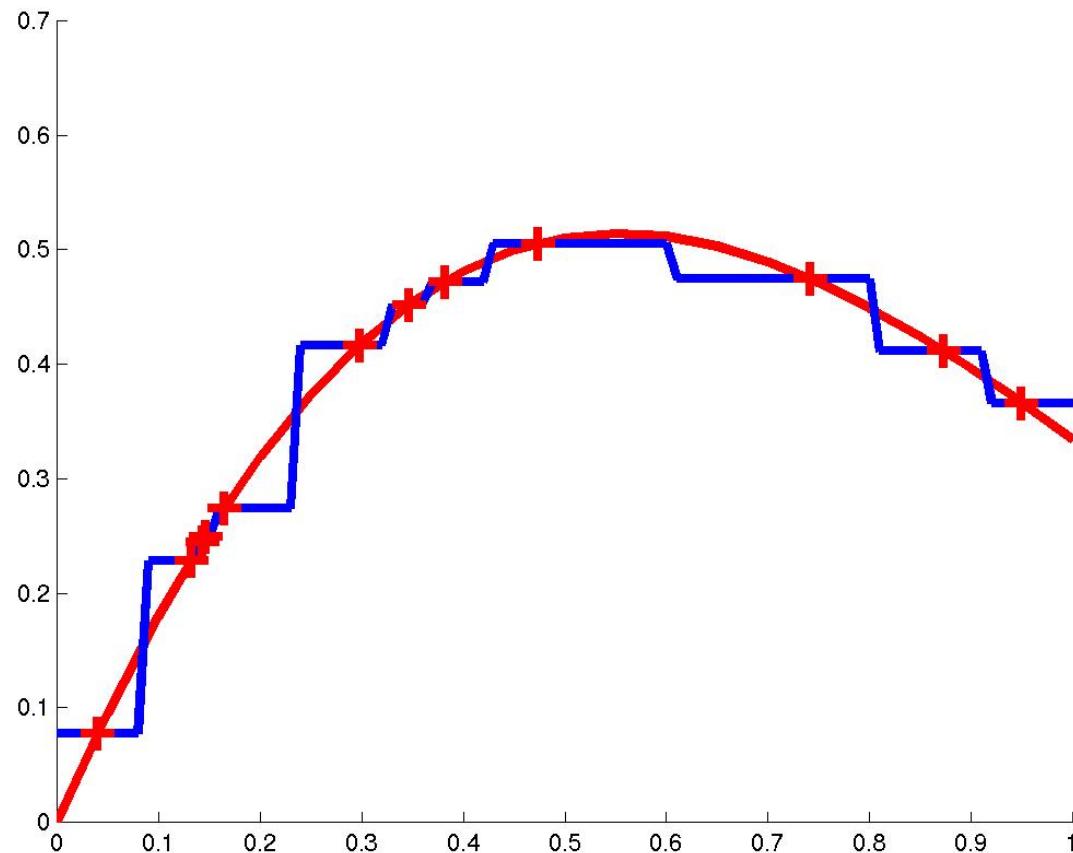


7-Nearest Neighbors



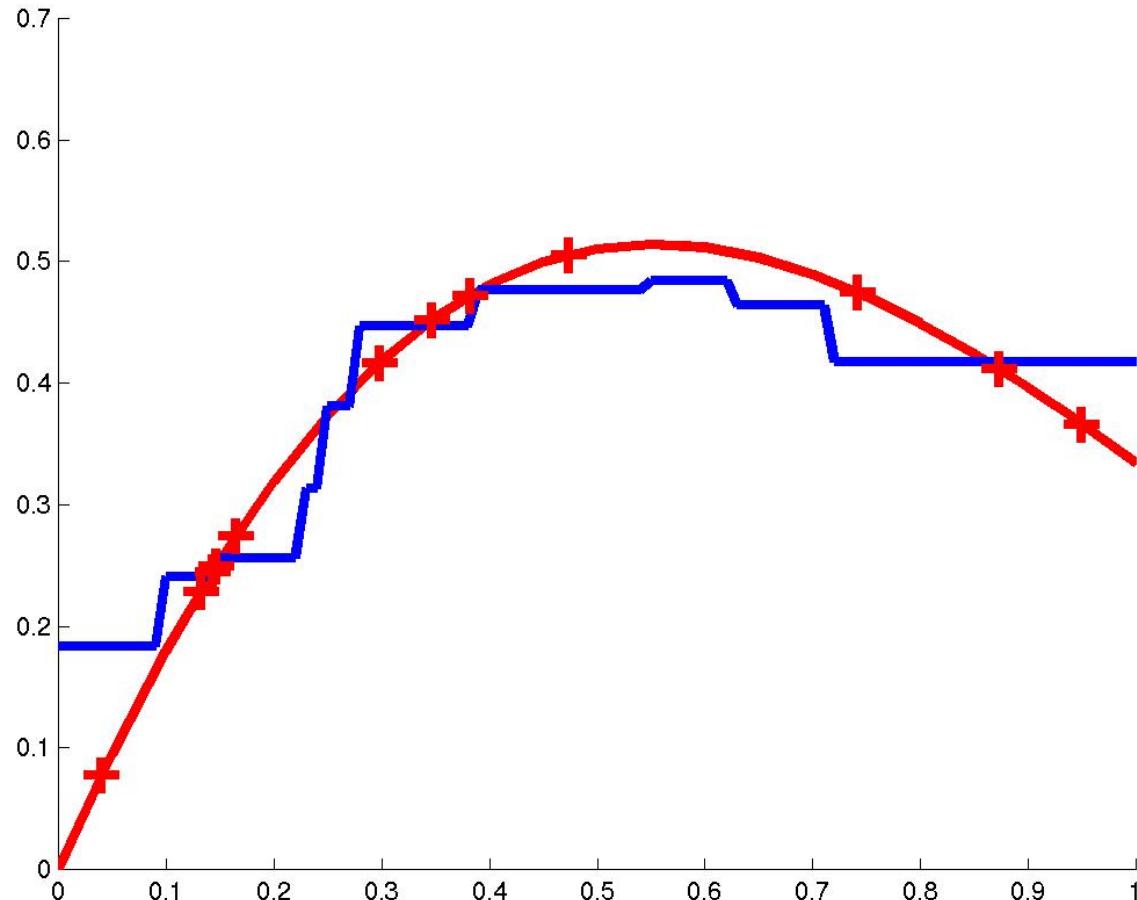
Nearest Neighbor (continuous)

1-nearest neighbor



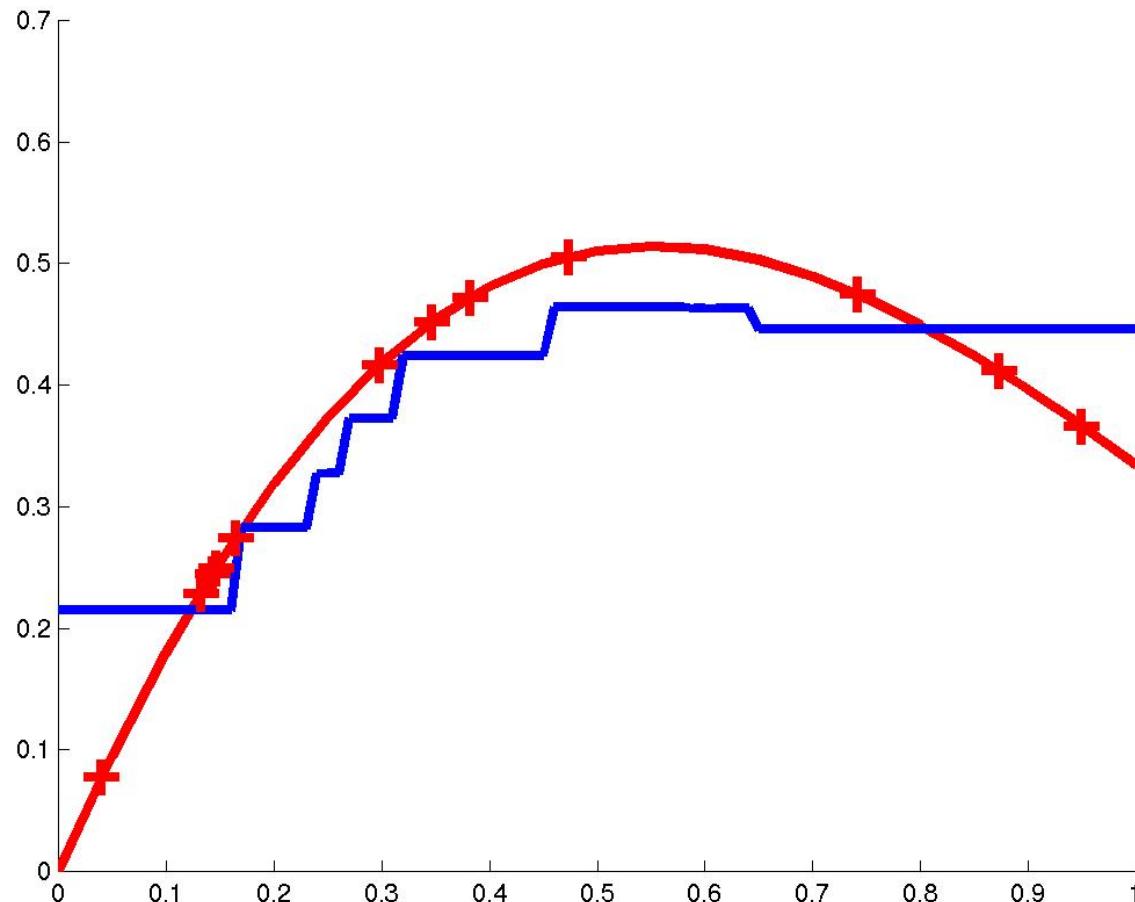
Nearest Neighbor (continuous)

3-nearest neighbor



Nearest Neighbor (continuous)

5-nearest neighbor



Distance Measures

- Euclidean distance
- Hamming distance
- City-Block distance
- Square distance
- Manhattan distance
- ...

Curse of Dimensionality

Imagine instances described by 20 attributes but only 2 are relevant to target function

Curse of dimensionality: nearest neighbor is easily misled when instance space is high-dimensional

One approach:

- Stretch j -th axis by weight z_j , where z_1, \dots, z_n chosen to minimize prediction error
- Use cross-validation to automatically choose weights z_1, \dots, z_n
- Note setting z_j to zero eliminates this dimension altogether (feature subset selection)

Lazy and Eager Learning

- Lazy: wait for query before generalizing
 - k-nearest neighbors, weighted linear regression
- Eager: generalize before seeing query
 - Radial basis function networks, decision trees, back-propagation
- Eager learner must create global approximation
- Lazy learner can create local approximations
- If they use the same hypothesis space, lazy can represent more complex functions ($H = \text{linear functions}$)

Reading Assignment

- What are the strengths of the k-NN
- What are the limitations of the k-NN
- What is Case based reasoning? How does it work?